

1 2 3

16/RTS

JC95 P

PCT/PTO

09/890816

01 AUG 2007

WO 00/45333

PCT/GB00/00277

1

1 "Neural Processing Element for use in a Neural Network"

2

3 The present invention relates to a neural processing
4 element for use in a neural network (NN). Particularly,
5 but not exclusively, the invention relates to a
6 scalable implementation of a modular NN and a method of
7 training thereof. More particularly, a hardware
8 implementation of a scalable modular NN is provided.

9

10 Artificial Neural Networks (ANNs) are parallel
11 information processing systems, whose parallelism is
12 dependent not only on the underlying
13 architecture/technology but also the algorithm and
14 sometimes on the intended application itself.

15

16 When implementing ANNs in hardware difficulties are
17 encountered as network size increases. The underlying
18 reasons for this are silicon area, pin out
19 considerations and inter-processor communications. One
20 aspect of the invention seeks to provide a scalable ANN
21 device comprising a modular system implemented on a
22 chip which seeks to mitigate or obviate the
23 difficulties encountered as the required network size
24 on the device increases. By utilising a modular
25 approach towards implementation, it is possible to
26 adopt a partitioning strategy to overcome the usual

1 The basic neuron does very little computation on its
2 own but when large numbers of neurons are used, the
3 total computation is often such that even the fastest
4 of serial computers is unable to train a network in a
5 reasonable time scale. The problem is exacerbated
6 because, the larger the network, the more training
7 steps are required and, consequently, the amount of
8 computation required increases exponentially with
9 increasing network size. There is also the added
10 problem of inter-neuron communication, which also
11 increases with increasing network size and must be
12 taken into account when attempting to implement
13 networks on parallel systems, because this
14 communication can become a bottleneck, preventing
15 substantial speedups for parallel implementations.

16
17 When considering parallel implementation of ANNs, it is
18 important to consider how the system is to be
19 parallelised. This is dependent not only on the
20 underlying architecture/technology but also the
21 algorithm and sometimes on the intended application
22 itself. However, there is often more than one approach
23 for any particular architecture and an understanding of
24 the consequences of partitioning strategies is of great
25 value. When using multi-processor systems, there are
26 two basic approaches to parallelising the Self-
27 Organising Map (SOM) algorithm; either the
28 functionality of the network can be partitioned such
29 that one processor may perform only one aspect of the
30 functionality of a neuron but performs this function
31 for a large number of neurons, or the network can be
32 partitioned so that a set of neurons (a set typically
33 consists of one or more neurons) is implemented on each
34 processor in the system.

35
36 Partitioning functionality of the network is an

1 approach that has been used with transputer systems
2 and, normally results in an architecture known as a
3 systolic array. The basic principle of the systolic
4 array is that the traditional single processing element
5 is replaced by an array of processing elements with
6 inputs and outputs only occurring at each end of the
7 array. The processing that would traditionally be
8 carried out by a single processor is then divided
9 amongst the processor array. Normally, each processor
10 would perform some of the functionality of the network
11 and that function would only be performed by that
12 processor. The array then acts as a pipeline of
13 processors, with data flowing in at one end and results
14 flowing out of the other. Unfortunately, this approach
15 is generally only appropriate for moderately sized
16 networks because the inter-processor communication
17 overheads become unmanageable very quickly and adding
18 more processors does little or nothing to alleviate the
19 problem.

20
21 When partitioning the SOM wherein one or more neurons
22 are implemented on an individual processor, the
23 communication overhead is lessened when compared to
24 approaches that partition functionality but can still
25 become a bottleneck as network size increases. Coarse
26 grain parallelism is the term generally associated with
27 a number of neurons implemented on each processor
28 whereas fine grain parallelism is the term used when
29 only a single neuron is implemented on individual
30 processors. The communication overhead tends to become
31 more prominent as the number of neurons per processor
32 is reduced because traditional processors are
33 implemented on separate devices and communication
34 between devices has much greater overheads than
35 communication amongst neurons on the same device. Fine
36 grain parallelism normally results in a Single

1 Instruction stream Multiple Data stream (SIMD) system
2 and is suited to massively parallel architectures such
3 as the Connection Machine.

4
5 If the implementation medium is to be in hardware such
6 as very large scale integration (VLSI) or similar, then
7 it may be possible to increase the level of parallelism
8 to the extent of implementing each weight in parallel.
9 However, this approach does little to improve overall
10 parallelism of the system because only part of the
11 functionality is performed at the weight level and
12 consequently, such an approach does not lead to the
13 most effective use of resources. The approach adopted
14 is fine grain parallelism with a single processing
15 element performing the functionality of a single
16 neuron. To overcome some of the inter-processor
17 communication problems it is suggested that several
18 processors be implemented on a single device with
19 strong short range communications.

20

21 Neural Network Implementations

22

23 In an attempt to overcome the limitations of general
24 purpose parallel computing platforms some researchers
25 attempted to develop specialised neural network
26 computers. Such approaches attempt to develop
27 architectures best suited to neural networks but are
28 normally based on the traditional parallel
29 architectures listed above. Modifications to these
30 basic architectural approaches have often been used in
31 an attempt to overcome some of the traditional problems
32 such as inter-processor communication. Others have
33 attempted to modify existing parallel systems such as
34 the Connection Machine to improve their usefulness as
35 neurocomputing architectures. Some have even
36 considered reconfigurable neurocomputer systems based

1 on Field Programmable Gate Array Technology (FPGA) but
2 most neurocomputer systems, while useful for
3 investigating the possibilities of ANNs, are normally
4 too large and expensive to be used for many
5 applications.

6
7 Driven mainly by the application domain researchers
8 undertook to investigate direct hardware implementation
9 of ANNs, and as biological neural systems appear to be
10 analogue, there was a bias towards analogue
11 implementation. Indeed, analogue implementation of
12 ANNs appears to be beneficial in some ways, e.g. very
13 little hardware is required for the memory elements of
14 such a system. However, there are also many problems
15 with analogue implementation of ANNs because the
16 fundamental building block of such systems is the
17 capacitor. Due to the shortcomings of the capacitor,
18 such as its tendency to suffer from leakage, a variety
19 of schemes were developed to overcome these weaknesses.

20
21 Macq et al proposed an analogue approach to
22 implementation of the SOM based on the use of currents
23 to represent weight values. Such an approach may
24 provide a mechanism for generating high density
25 integration due to the small number of transistors
26 required for each neuron, but this approach uses
27 analogue synaptic weights based on current copiers, the
28 principle component of which is the capacitor, which is
29 prone to leakage. These leakage currents continuously
30 modify the value stored by the capacitor thereby
31 necessitating some form of refreshment to maintain
32 reasonable precision of weight values. The main cause
33 of this leakage is the reverse biased junction. Their
34 proposed method of refreshment uses a converter to
35 periodically refresh each synaptic weight. This is
36 achieved by reading the current memorised by each cell

1 using successive approximation and then writing back to
2 the cell the next upper reference current. It is
3 claimed that this approach allows for on chip learning.
4 However, for the gain factor to reduce with time, as
5 prescribed by Kohonen, adjustments need to be made to
6 the reset signal, and for the neighbourhood to reduce
7 with time the period of one of the timing circuit
8 clocks must be adjusted. The impression given is that
9 these changes would require manual intervention. The
10 leakage current of capacitors also appears to be the
11 main factor that would restrict the maximum number of
12 memory cells in this design.

13
14 A charge based approach to implementation was suggested
15 in "A Charge-Based On-Chip Adaptation Kohonen Neural
16 Network" which claims that such an approach would lead
17 to low power dissipation and compact device
18 configurations. The approach uses switched capacitor
19 circuits to store the weights and the adaptive weight
20 synapses used utilises parasitic capacitances between
21 two adjacent gates of the switched capacitor circuit to
22 determine the learning rate. This will give a fixed
23 learning rate, which will be different for each device
24 manufactured due to the difficulties in manufacturing
25 such components to exactly the same parameters from
26 device to device. Weight integrity is also a potential
27 problem area because, as with most analogue
28 implementations of neural networks, weight values are
29 stored by capacitors which have difficulty maintaining
30 the charge held, and consequently the weight value.
31 The authors of this paper attempt to address this issue
32 but, for weights not being updated during a cycle, they
33 simply regarded it as a forget effect. Unfortunately,
34 as the number of neurons on the device increases, so
35 too does the common node parasitic capacitance. This
36 will require the size of the storage electrode of each

1 neuron to be increased as network size increases to
2 compensate.

3
4 Perhaps the most successful analogue implementations
5 are those which utilise a pulse stream approach. It
6 has long been known that biological neural systems use
7 pulses to communicate between cells and simple
8 oscillating circuits can be implemented in VLSI
9 relatively easily. Unfortunately, the problem of
10 analogue memory still overshadows such approaches. The
11 main advantage of pulse stream approaches is that
12 hardware requirements for the arithmetic units are very
13 low compared to the equivalent digital implementation;
14 in particular multipliers which can be implemented in
15 an analogue fashion using only three transistors
16 require many gates for digital systems.

17
18 The problems of implementing digital multipliers and
19 storing weight values provide two reasons that most
20 digital implementations of the SOM have been restricted
21 to small network sizes and are often only coprocessors
22 rather than fully parallel implementations. The other
23 main factor that has made a significant contribution to
24 limiting network size is the inter-neuron communication
25 overhead which increases exponentially with network
26 size. Consequently, most fully digital implementations
27 of the SOM require some modification to Kohonen's
28 original algorithm, e.g. Ienne et al suggest two
29 alternative modifications to the SOM algorithm for
30 digital implementation. Van den Bout et al also
31 propose an all digital implementation of the SOM and
32 investigate a rapid prototyping approach towards neural
33 network hardware development. This is facilitated by
34 the use of Xilinx field programmable gate arrays
35 (FPGAs) which provide a flexible platform for such
36 endeavours and speed up construction time compared to

1 VLSI development. Their approach uses stochastic
2 signals to allow pseudo-analogue computation to be
3 carried out using space efficient digital logic. A
4 Markovian learning algorithm is used to simplify that
5 suggested by Kohonen and the Manhattan distance metric
6 is used in place of Euclidean distance to simplify
7 distance calculations. Their approach towards the
8 implementation of the SOM is later reiterated when they
9 describe their VLSI implementation, TInMann.

10

11 Saarinen et al propose a fully digital approach to the
12 implementation of Kohonen's SOM in order to create a
13 neural coprocessor for PC based systems. Their
14 approach uses three Xilinx XC3090 FPGAs to create 16
15 processing elements, and RAM to store both weight and
16 input vector values. The host computer initialises the
17 random weight values, loads up the input vector values
18 and sets the network parameters (i.e. network size,
19 number of inputs, gain factor and number of training
20 steps). After the host computer has set these
21 parameters the coprocessor system then trains the
22 network according to the pre-specified parameters until
23 training is complete. The architecture of the system
24 consists of three main elements; a distance and update
25 unit (DUU), a distance comparator unit (DCU) and an
26 address control unit (ACU), each implemented on a
27 separate FPGA which is clearly a partitioning of the
28 network functionality and is not likely to be scaleable
29 due to the communication overheads. In addition, this
30 implementation does not implement the standard SOM but,
31 a rather limited, one dimensional version.

32

33 While more obvious than many of the digital
34 implementation approaches used, that of Saarinen is
35 rather typical in that it partitions functionality.
36 Most digital implementations appear to do the same, but

1 they maintain the whole system on a single device. The
2 rationale behind this is that when using digital
3 multipliers, vast resources are normally required to
4 implement them, so it is often more effective to have a
5 limited number but to make them fast. To avoid using
6 excessive resources for the Modular Map implementation,
7 very limited reduced instruction set computers (RISC)
8 processors are suggested that use an alternative
9 approach to multiplication which will only require a
10 fraction of the resources needed to implement a
11 traditional digital multiplier. In addition, while
12 minor modifications to Kohonen's algorithm are made,
13 its basic operation and two dimensional nature are
14 maintained.

15
16 The paper by Ruping et al presented simultaneously with
17 the paper by Lightowler et al presents a fully digital
18 hardware implementation of the SOM which incorporates
19 some of the same ideas as does the Modular Map design.
20 To facilitate hardware implementation Ruping et al also
21 use Manhattan distance instead of Euclidean distance
22 and the gain factor is restricted to negative powers of
23 two. A system comprising 16 devices is outlined and
24 performance information is presented in terms of the
25 operating speed of the system etc. Each of their
26 devices implements 25 neurons as separate processing
27 elements and allows for network size to be increased by
28 using several devices. However, these devices only
29 contain neurons; there is no local control for the
30 neurons on a device. An external controller is
31 required to interface with these devices and control
32 the actions of their constituent neurons.
33 Consequently, these devices are not autonomous as are
34 Modular Maps and only lateral expansion which creates a
35 Single Instruction stream Multiple Data stream (SIMD)
36 architecture has been considered as an approach towards

1 creating larger network sizes.

2

3 There have also been some commercial hardware
4 implementations of ANNs, the number of which has been
5 steadily growing over the last few years. They
6 generally offer a speedup of around an order of
7 magnitude compared to implementation on a PC alone but
8 are predominantly coprocessors rather than stand alone
9 systems and are not normally scaleable. However, while
10 some of these implementations are only able to
11 implement a single ANN paradigm, most use digital
12 signal processing (DSP) chips, transputers or standard
13 microprocessors, thereby allowing the system to be
14 programmable to some extent and implement a range of
15 standard ANNs.

16

17 The commercially available approach to implementation,
18 (i.e. accelerator cards) offers the slowest speedup of
19 the main implementation approaches but can still offer
20 a significant speedup compared to simulation on
21 standard PC systems and the growing number available on
22 the market suggests that they are useful for a range of
23 applications. General purpose multiprocessor systems
24 offer a further speedup but large scale systems
25 normally have significant communication overheads.
26 Some researchers have attempted to modify standard
27 multiprocessor architectures to improve their
28 application to ANNs and have increased achievable
29 speedup by doing so but while these systems have been
30 useful in ANN research, they are not fully scaleable
31 and require significant financial outlay. The greatest
32 speedups for ANN implementations have been achieved by
33 dedicated neural network chips but the problem again
34 has been that these systems are limited to relatively
35 small scale systems. As an approach towards developing
36 scaleable neural network systems, there have been some

1 attempts at developing modular systems.

2

3 Modular Systems

4

5 There is considerable evidence to suggest that
6 biological neural systems have a modular organisation
7 at various levels. At a macroscopic level, for
8 example, it has been found that some people have no
9 connection between the left and right hemispheres of
10 the brain, which does bring with it certain problems,
11 but they are still able to function in a near to normal
12 way, which shows that each hemisphere is able to
13 function independently. However, it has also been
14 noted that, while each hemisphere is almost identical
15 physiologically, they specialise in functionality.
16 When one begins to look closer at the cerebral
17 hemisphere one finds that different functionality is
18 found at different regions, even though these regions
19 show a modular organisation and are made up of
20 geometrically defined repetitive units. Research by
21 Murre and Sturdy also supports this view of a modular
22 organisation in their attempt at a quantitative
23 analysis of the brain's connectivity. It is of
24 interest that this modularity is also seen in relation
25 to the topological maps formed in the neo-cortex, e.g.
26 somatosensory maps for different parts of the body are
27 found at different parts of the cerebral cortex and
28 similar maps for other senses such as sound (tonotopic
29 maps) are found in different regions again. Such
30 evidence suggests that while the concept of topological
31 maps which form the basis for Kohonen's self organising
32 map is valid, it also suggests that the brain contains
33 many of these maps. Consequently, it is reasonable to
34 suggest that when attempting to develop scaleable, and
35 particularly when trying to develop large scale
36 implementations of the SOM, that a modular approach

1 should be considered.

2
3 Researchers such as Happel and Murre have approached
4 neural network design as an evolutionary process using
5 genetic algorithms to determine network architectures.
6 Their investigations into the design of modular neural
7 networks using the CALM module are intended as a study
8 to assist with understanding of the relationship
9 between structure and functionality in the brain but
10 they present some findings that may also assist with
11 the development of information processing systems.
12 They found that the best performing network
13 architectures derived with their approach reproduced
14 characteristics of the vision system with the
15 organisation of coarse and fine processing of stimuli
16 in different pathways. They also present a range of
17 evidence that supports the belief that the brain is
18 highly organised and modular in its architecture.

19
20 The basic premise on which modular neural network
21 systems are developed is that the computation performed
22 by the network is decomposed into two or more separate
23 modules which operate as individual entities. Not only
24 can such approaches improve scaleability but
25 considerable savings can be made on the learning times
26 required for large networks, which are often rather
27 slow. In addition, the generalisation abilities of
28 large networks are often poor, whereas systems composed
29 of several modules do not appear to suffer from this
30 drawback. Research carried out by Jacobs et al using
31 modules composed of Multi Layer Perceptrons (MLPs) used
32 competition to split the input space into overlapping
33 regions. Their work found that the modular approach
34 had much improved training times compared to single
35 large networks and gave better performance, especially
36 where there were discontinuities within classes in the

1 original input space. They also found, when building
2 hierarchies of such systems, an architecture they refer
3 to as a hierarchical mixture of experts, that the
4 results yielded a probabilistic approach to decision
5 tree modelling. Others, such as Hansen and Salamon,
6 have considered ensembles of neural networks as a means
7 of improving classification. Essentially the ensemble
8 approach involves training several networks on the same
9 task to achieve a more reliable output.

10

11 A modular approach to implementation of the SOM is a
12 valid alternative to the more traditional approaches
13 which attempt to create single networks. Other authors
14 such as Helge Ritter have also presented research
15 supporting a modular approach for the SOM. There also
16 appears to be a sound basis for modularity in
17 biological systems and, while no attempt is being made
18 to replicate biological systems, they are nevertheless
19 the initial inspiration for artificial neural networks.
20 It is also pertinent to consider that, while Man has
21 only been attempting to develop computing systems for a
22 matter of centuries, natural evolution had produced a
23 range of biological computers long before Man was on
24 this earth. Even with the latest of modern technology,
25 Man is unable to create computers that surpass the
26 computing abilities of biological systems, so it is
27 suggested that Man should continue to learn from
28 nature.

29

30 According to a first aspect of the present invention,
31 there is provided a neuron for use in a neural network,
32 the neuron comprising

33 an arithmetic logic unit;
34 a shifter mechanism;
35 a set of registers;
36 an input port;

1 an output port; and
2 control logic.

3

4 According to a second aspect of the present invention,
5 there is provided a module controller for controlling
6 the operation of at least one neuron, the controller
7 comprising

8 an input port;
9 an output port;
10 a programmable read-only memory;
11 an address map;
12 an input buffer; and
13 at least one handshake mechanism.

14

15 According to a third aspect of the present invention,
16 there is provided a neuron module, the module
17 comprising

18 at least one neuron; and
19 at least one module controller.

20

21 Preferably, the at least one neuron and the at least
22 one module controller are implemented on one device.
23 The device is typically a field programmable gate array
24 (FPGA) device. Alternatively, the device may be a
25 full-custom very large scale integration (VLSI) device,
26 a semi-custom VLSI or an application specific
27 integrated circuit (ASIC).

28

29 According to a fourth aspect of the present invention
30 there is provided a neural network, the network
31 comprising

32 at least two neuron modules coupled together.

33

34 Typically, the neuron modules are coupled in a lateral
35 expansion mode. Alternatively, the neuron modules may
36 be coupled in a hierarchical mode. Optionally, the

1 neuron modules may be coupled in a combination of
2 lateral expansion modes and hierarchical modes.

3
4 In lateral expansion mode, the at least two neuron
5 modules are typically connected on a single plane.
6 Data is preferably input to the modules in the network
7 only once. Thus, the modules forming the network are
8 synchronised to facilitate this. The modules are
9 preferably synchronised using a two-line handshake
10 mechanism. The two-line mechanism typically has two
11 states. The two states typically comprise a wait state
12 and a data ready state. The wait state typically
13 occurs where a sender and/or a receiver is not ready
14 for the transfer of data from the sender to the
15 receiver or vice versa. The data ready state typically
16 occurs when both the sender and receiver are ready for
17 data transfer. Data transfer follows immediately the
18 data ready state occurs.

19
20 The neuron modules typically comprise at least one
21 neuron, and at least one module controller.

22
23 Typically, the number of neurons in a module is a power
24 of two. The number of neurons in a module is
25 preferably 256. Any number of neurons may be used in a
26 module, but the number of neurons is preferably a power
27 of two.

28
29 A neuron typically comprises an arithmetic logic unit,
30 a shifter mechanism, a set of registers, an input port,
31 an output port, and control logic.

32
33 The arithmetic logic unit (ALU) typically comprises an
34 adder/subtractor unit. The ALU is typically at least a
35 4-bit adder/subtractor unit, and preferably a 12-bit
36 adder/subtractor unit. The adder/subtractor unit

1 typically includes a carry lookahead adder (CLA).
2
3 The ALU typically includes at least two flags. A zero
4 flag is typically set when the result of an arithmetic
5 operation is zero. A negative flag is typically set
6 when the result of an arithmetic operation is negative.
7
8 The ALU typically further includes at least two
9 registers. A first register is typically located at
10 one of the inputs to the ALU. A second register is
11 typically located at the output from the ALU. The
12 second register is typically used to store data until
13 it is ready to be transferred eg stored.
14
15 The shifter mechanism typically comprises an arithmetic
16 shifter. The arithmetic shifter is typically
17 implemented using flip-flops. The shifter mechanism is
18 preferably located in a data stream between the output
19 of the ALU and the second register of the ALU. This
20 location increases the flexibility of the neuron and
21 increases the simplicity of the design.
22
23 The control logic typically comprises a reduced
24 instruction set computer (RISC). The instruction set
25 typically comprises thirteen different instructions.
26
27 The module controller typically comprises an input
28 port, an output port, a programmable read-only memory,
29 an address map, an input buffer, and at least one
30 handshake mechanism.
31
32 The programmable read-only memory (PROM) typically
33 contains the instructions for the controller and/or the
34 subroutines for the at least one neuron.
35
36 The address map typically allows for conversion between

1 a real address and a virtual address of the at least
2 one neuron. The real address is typically the address
3 of a neuron on the device. The virtual address is
4 typically the address of the neuron within the network.
5 The virtual address is typically two 8-bit values
6 corresponding to X and Y co-ordinates of the neuron on
7 the single plane.

8
9 The at least one handshake mechanism typically includes
10 a synchronisation handshake mechanism for synchronising
11 data transfer between a sender and a receiver module.
12 The synchronisation handshake mechanism typically
13 comprises a three-line mechanism. The three-line
14 mechanism typically has three states. The three states
15 typically comprise a wait state, a no device state and
16 a data ready state. The wait state typically occurs
17 where a sender and/or a receiver is not ready for the
18 transfer of data from the sender to the receiver or
19 vice versa. The no device state is typically used
20 where inputs are not present. Thus, reduced input
21 vector sizes may be used. The no device state may also
22 be used to prevent the controller from malfunctioning
23 when an input stream(s) is temporarily lost or stopped.
24 The data ready state typically occurs when both the
25 sender and receiver are ready for data transfer. Data
26 transfer follows immediately when the data ready state
27 occurs. The three-line mechanism typically comprises
28 two outputs from the receiver and one output from the
29 sender. The advantage of the three-line mechanism is
30 that no other device is required to facilitate data
31 transmission between the sender and receiver or vice
32 versa. Thus, the transmission of data is directly from
33 point to point.

34
35 According to a fifth aspect of the present invention
36 there is provided a method of training a neural

1 network, the method comprising the steps of
2 providing a network of neurons;
3 reading an input vector applied to the input of
4 the neural network;
5 calculating the distance between the input vector
6 and a reference vector for all neurons in the network;
7 finding the active neuron;
8 outputting the location of the active neuron; and
9 updating the reference vectors for all neurons in
10 a neighbourhood around the active neuron.

11
12 A distance metric is typically used to calculate the
13 distance between the input vector and the reference
14 vector. Preferably, the Manhattan distance metric is
15 used. Alternatively, a Euclidean distance metric may
16 be used.

17
18 Calculation of the Manhattan distance preferably uses a
19 gain factor. The value of the gain factor is
20 preferably restricted to negative powers of two.

21
22 The network of neurons typically comprises a neural
23 network. The neural network typically comprises at
24 least two neuron modules coupled together.

25
26 Typically, the neuron modules are coupled in a lateral
27 expansion mode. Alternatively, the neuron modules may
28 be coupled in a hierarchical mode. Optionally, the
29 neuron modules may be coupled in a combination of
30 lateral expansion modes and hierarchical modes.

31
32 In lateral expansion mode, the at least two neuron
33 modules are typically connected on a single plane.
34 Data is preferably input to the modules in the network
35 only once. Thus, the modules forming the network are
36 synchronised to facilitate this. The modules are

1 preferably synchronised using a two-line handshake
2 mechanism. The two-line mechanism typically has two
3 states. The two states typically comprise a wait state
4 and a data ready state. The wait state typically
5 occurs where the sender and/or the receiver is not
6 ready for the transfer of data from the sender to the
7 receiver or vice versa. The data ready state typically
8 occurs when both the sender and receiver are ready for
9 data transfer. Data transfer follows immediately the
10 data ready state occurs.

11

12 The neuron modules typically comprise at least one
13 neuron, and at least one module controller.

14

15 Preferably, the at least one neuron and the at least
16 one module controller are implemented on one device.
17 The device is typically a field programmable gate array
18 (FPGA) device. Alternatively, the device may be a
19 full-custom very large scale integration (VLSI) device,
20 a semi-custom VLSI or an application specific
21 integrated circuit (ASIC).

22

23 Typically, the number of neurons in a module is a power
24 of two. The number of neurons in a module is
25 preferably 256. Any number of neurons may be used in a
26 module, but the number of neurons is preferably a power
27 of two.

28

29 A neuron typically comprises an arithmetic logic unit,
30 a shifter mechanism, a set of registers, an input port,
31 an output port, and control logic.

32

33 The arithmetic logic unit (ALU) typically comprises an
34 adder/subtractor unit. The ALU is typically at least a
35 4-bit adder/subtractor unit, and preferably a 12-bit
36 adder/subtractor unit. The adder/subtractor unit

1 typically includes a carry lookahead Adder (CLA).

2

3 The ALU typically includes at least two flags. A zero
4 flag is typically set when the result of an arithmetic
5 operation is zero. A negative flag is typically set
6 when the result of an arithmetic operation is negative.

7

8 The ALU typically further includes at least two
9 registers. A first register is typically located at
10 one of the inputs to the ALU. A second register is
11 typically located at the output from the ALU. The
12 second register is typically used to store data until
13 it is ready to be transferred eg stored.

14

15 The shifter mechanism typically comprises an arithmetic
16 shifter. The arithmetic shifter is typically
17 implemented using flip-flops. The shifter mechanism is
18 preferably located in a data stream between the output
19 of the ALU and the second register of the ALU. This
20 location increases the flexibility of the neuron and
21 increases the simplicity of the design.

22

23 The control logic typically comprises a reduced
24 instruction set computer (RISC). The instruction set
25 typically comprises thirteen different instructions.

26

27 The module controller typically comprises an input
28 port, an output port, a programmable read-only memory,
29 an address map, an input buffer, and at least one
30 handshake mechanism.

31

32 The programmable read-only memory (PROM) typically
33 contains the instructions for the controller and/or the
34 subroutines for the at least one neuron.

35

36 The address map typically allows for conversion between

1 a real address and a virtual address of the at least
2 one neuron. The real address is typically the address
3 of a neuron on the device. The virtual address is
4 typically the address of the neuron within the network.
5 The virtual address is typically two 8-bit values
6 corresponding to X and Y co-ordinates of the neuron on
7 the single plane.

8
9 The at least one handshake mechanism typically includes
10 a synchronisation handshake mechanism for synchronising
11 data transfer between a sender and receiver module.
12 The synchronisation handshake mechanism typically
13 comprises a three-line mechanism. The three-line
14 mechanism typically has three states. The three states
15 typically comprise a wait state, a no device state and
16 a data ready state. The wait state typically occurs
17 where the sender and/or the receiver is not ready for
18 the transfer of data from the sender to the receiver or
19 vice versa. The no device state is typically used
20 where inputs are not present. Thus, reduced input
21 vector sizes may be used. The no device state may also
22 be used to prevent the controller from malfunctioning
23 when an input stream(s) is temporarily lost or stopped.
24 The data ready state typically occurs when both the
25 sender and receiver are ready for data transfer. Data
26 transfer follows immediately when the data ready state
27 occurs. The three-line mechanism typically comprises
28 two outputs from the receiver and one output from the
29 sender. The advantage of the three-line mechanism is
30 that no other device is required to facilitate data
31 transmission between the sender and receiver or vice
32 versa. Thus, the transmission of data is directly from
33 point to point.

34
35
36

1 Embodiments of the present invention shall now be
2 described, with reference to the accompanying drawings
3 in which:-

4 Fig. 1a is a unit circle for a Euclidean distance
5 metric;
6 Fig. 1b is a unit circle for a Manhattan distance
7 metric;
8 Fig. 2 is a graph of gain factor against training
9 time;
10 Fig. 3 is a diagram showing neighbourhood
11 function;
12 Figs 4a-c are examples used to illustrate an
13 elastic net principle;
14 Fig. 5 is a schematic diagram of a single Modular
15 Map;
16 Fig. 6 is a schematic diagram of laterally
17 combined Maps;
18 Fig. 7 is a schematic diagram of hierarchically
19 combined Maps;
20 Fig. 8 is a scatter graph showing input data
21 supplied to the network of Fig. 7;
22 Fig. 9 is a Voronoi diagram of a module in an
23 input layer I of Fig. 7;
24 Fig. 10 is a diagram of input layer activation
25 regions for a level 2 module with 8 inputs;
26 Fig. 11 is a schematic diagram of a Reduced
27 Instruction Set Computer (RISC) neuron;
28 Fig. 12 is a schematic diagram of a module
29 controller system;
30 Fig. 13 is a state diagram for a three-line
31 handshake mechanism;
32 Fig. 14 is a flowchart showing the main processes
33 involved in training a neural network;
34 Fig. 15 is a graph of activations against training
35 steps for a typical neural net;
36 Fig. 16 is a graph of training time against

1 network size using 16 and 99 element reference
2 vectors;
3 Fig. 17 is a log-linear plot of relative training
4 times for different implementation strategies for
5 a fixed input vector size of 128 elements;
6 Fig. 18 is example greyscale representation of the
7 range of images for a single subject used in a
8 human face recognition application;
9 Fig. 19a is an example activation pattern created
10 by the same class of data for a modular map shown
11 in Fig. 23;
12 Fig. 19b is an example activation pattern created
13 by the same class of data for a 256 neuron self-
14 organising map (SOM);
15 Fig. 20 is a schematic diagram of a modular map
16 (configuration 1);
17 Fig. 21 is a schematic diagram of a modular map
18 (configuration 2);
19 Fig. 22 is a schematic diagram of a modular map
20 (configuration 3);
21 Fig. 23 is a schematic diagram of a modular map
22 (configuration 4);
23 Figs 24a to 24e are average time domain signals
24 for a 10kN, 20kN, 30kN, 40kN and blind ground
25 anchorage pre-stress level tests, respectively;
26 Figs 25a to 25e are average power spectrum for the
27 time domain signals in Figs 24a to 24e
28 respectively;
29 Fig. 26 is an activation map for a SOM trained
30 with the ground anchorage power spectra of Figs
31 25a to 25e;
32 Fig. 27 is a schematic diagram of a modular map
33 (configuration 5);
34 Fig. 28 is the activation map for module 0 in Fig.
35 27;
36

1 Fig. 29 is the activation map for module 1 in Fig.
2 27;
3 Fig. 30 is the activation map for module 2 in Fig.
4 27;
5 Fig. 31 is the activation map for module 3 in Fig.
6 27; and
7 Fig. 32 is the activation map for an output module
8 (module 4) in Fig. 27.

9
10 As an approach to overcoming the constraints of unitary
11 artificial neural networks a modular implementation
12 strategy for the Self-Organising Map (SOM) can be used.
13 The basic building block of this system is the Modular
14 Map which is itself a parallel implementation of the
15 SOM. Kohonen's original algorithm has been maintained,
16 excepting that parameters have been quantised and the
17 Euclidean distance metric used as standard has been
18 replaced by Manhattan distance. Each module contains
19 sufficient neurons to enable it to do useful work as a
20 stand alone system. However, the Modular Map design is
21 such that many modules can be connected together to
22 create a wide variety of configurations and network
23 sizes. This modular approach results in a scaleable
24 system that meets an increased workload with an
25 increase in parallelism and thereby avoids the usually
26 extensive increases in training times associated with
27 unitary implementations.

28 **Background**

30
31 An important premise on which the Modular Map has been
32 developed is its ability to form topological maps of
33 the input space, a phenomenon which has been likened to
34 the 'neuronal maps' of the brain which are found in
35 regions of the neo-cortex associated with various
36 senses. The formation of such topology preserving maps

1 occurs during the learning process defined for the Self
2 Organising Map (SOM).

3

4 In the Modular Map implementation of the SOM the
5 multidimensional Euclidean input space \mathbb{R}^n , where \mathbb{R}
6 covers the range (0, 255) and ($0 < n \leq 16$), is
7 mapped to a two dimensional output space \mathbb{R}^2 (where the
8 upper limit on \mathbb{R} is variable between 8 and 255) by way
9 of a non-linear projection of the probability density
10 function. Each neuron in the network has a reference
11 vector $m_i = [\mu_{i1}, \mu_{i2}, \dots, \mu_{in}] \in \mathbb{R}^n$ where μ_{ij} are
12 scalar weights, i is the neuron index and j the weight
13 index.

14

15 An input vector $x = [\epsilon_1, \epsilon_2, \dots, \epsilon_n] \in \mathbb{R}^n$ is presented
16 to all neurons in the network where the closest
17 matching reference vector (codebook vector) C
18 is determined, i.e.

$$19 \quad \sum_{j=0}^n |\xi_j - \mu_{cj}| = \min \left\{ \sum_{j=0}^n |\xi_j - \mu_{ij}| \right\}_{i=1}^k$$

21 where k = network size.

22

23 The neuron with minimum distance between its codebook
24 vector and the current input (i.e. greatest similarity)
25 becomes the active neuron. A variety of distance
26 metrics can be used as a measure of similarity, the
27 Euclidean distance being the most popular. However, it
28 should be noted that the distance metric being used
29 here is Manhattan distance, known to many as the
30 L_1 metric of the family of Minkowski metrics, i.e.
31 the distance between two points a and b is

32

$$33 \quad L_p = (|a - b|^p + |a - b|^p)^{1/p}$$

34

35 Clearly, Euclidean distance would be the L_2 metric
36 under Minkowski's scheme. An idea of these two

1 distance functions can be gained by plotting the unit
2 circle for both metrics. Fig 1a shows the unit circle
3 for the Euclidean metric, and Fig. 1b shows the unit
4 circle for the Manhattan metric.

5
6 The Manhattan distance metric is both simple to
7 implement and a reasonable alternative to the Euclidean
8 distance metric which is rather expensive to implement
9 in terms of hardware due to the need to calculate
10 squares of the distances involved.

11
12 After the active neuron has been identified reference
13 vectors are updated to bring them closer to the current
14 input vector. The amount by which codebook vectors are
15 changed is determined by their distance from the input,
16 and the current gain factor $\alpha(t)$. If neurons are
17 within the neighbourhood of the active neuron then
18 their reference vectors are updated, otherwise no
19 changes are made.

20

$$21 \quad m_i(t+1) = m_i(t) + \alpha(t)[x(t) - m_i(t)] \text{ if } i \in N_c(t)$$

22

23

24 and

25

$$26 \quad m_i(t+1) = m_i(t) \text{ if } i \notin N_c(t)$$

27

28

29 where $N_c(t)$ is the current neighbourhood and $t = 0, 1,$
30 $2, \dots$

31

32 Both the gain factor and neighbourhood size decrease
33 with time from their original start-up values
34 throughout the training process. Due to implementation
35 considerations these parameters are constrained to a
36 range of discrete values rather than the continuum

1 suggested by Kohonen. However, the algorithms chosen
2 to calculate values for gain and neighbourhood size
3 facilitate convergence of codebook vectors in line with
4 Kohonen's original algorithm.

5
6 The gain factor $\alpha(t)$ being used by the Modular Map is
7 restricted to negative powers of two to simplify
8 implementation. Fig. 2 is a graph of gain factor $\alpha(t)$
9 against training time when the gain factor $\alpha(t)$ is
10 restricted to negative powers of two. By restricting
11 the gain factor $\alpha(t)$ in this way it is possible to use
12 a bit shift operation for multiplication rather than
13 requiring an additional hardware multiplier which would
14 clearly require more hardware and increase the
15 complexity of the implementation. This approach does
16 not unduly affect the performance of the algorithm and
17 is suitable for simplifying hardware requirements.

18
19 A square, step function neighbourhood, one of several
20 approaches suggested by Kohonen, could be defined by
21 the Manhattan distance metric. This approach to
22 defining the neighbourhood has the effect of rotating
23 the square through 45 degrees and can be used by
24 individual neurons to determine if they are in the
25 current neighbourhood when given the index of the
26 active neuron (see Fig. 3). Fig. 3 is a diagram
27 showing the neighbourhood function when a square, step
28 function neighbourhood is used. When all these
29 parameters are combined to form the Modular Map it has
30 the same characteristics as the self-organising map and
31 gives comparable results when evaluated. The
32 architecture of the Modular Map was also designed to
33 allow for expansion by combining many such modules
34 together to create larger maps while avoiding the usual
35 communications bottleneck and maintaining
36 self-organising map behaviour.

1 Stand Alone Maps

2
3 If, for visualisation purposes, a simplified case of
4 the Modular Map is considered where only three
5 dimensions are used as inputs, then a single map would
6 be able to represent an input space enclosed by a cube
7 and each dimension would have a possible range of
8 values between 0 and 255. With only the simplest of
9 pre-processing this cube could be placed anywhere in
10 the input space \mathbb{R}^n where \mathbb{R} covers the range $(-\infty$ to $+\infty)$,
11 and the codebook vector of each neuron within the
12 module would give the position of a point somewhere
13 within this feature space. The implementation
14 suggested would allow each vector element to hold
15 integer values within the given scale, so there are a
16 finite number of distinct points which can be
17 represented within the cube (i.e. 256^3). Each of the
18 points given by the codebook vectors has an 'elastic'
19 sort of bond between itself and the point denoted by
20 the codebook vectors of neighbouring neurons so as to
21 form an elastic net (Fig. 4).

22
23 Figs 4a to 4c shows a series of views of the elastic
24 net when an input is presented to the network. The
25 figures show the point position of reference vectors in
26 three dimensional Euclidean space along with their
27 elastic connections. For simplicity, reference vectors
28 are initially positioned in the plane with $z=0$, the
29 gain factor $\alpha(t)$ is held constant at 0.5 and both
30 orthogonal and plan views are shown. After the input
31 has been presented, the network proceeds to update
32 reference vectors of all neurons in the current
33 neighbourhood. In Fig. 4b, the neighbourhood function
34 has a value of three. In Fig. 4c the same input is
35 presented to the network for a second time and the
36 neighbourhood is reduced to two for this iteration.

1 Note that the reference points around the active neuron
2 become close together as if they were being pulled
3 towards the input by elastic bonds between them.

4
5 Inputs are presented to the network in the form of
6 multi-dimensional vectors denoting positions within the
7 feature space. When an input is received, all neurons
8 in the network calculate the similarity between their
9 codebook vectors and the input using the Manhattan
10 distance metric. The neuron with minimum Manhattan
11 distance between its codebook vector and the current
12 input, (i.e. greatest similarity) becomes the active
13 neuron. The active neuron then proceeds to bring its
14 codebook vector closer to the input, thereby increasing
15 their similarity. The extent of the change applied is
16 proportional to the distance involved, this
17 proportionality being determined by the gain factor
18 $\alpha(t)$, a time dependent parameter.

19
20 However, not only does the active neuron update its
21 codebook vector, so too do all neurons in the current
22 neighbourhood (i.e. neurons topographically close to
23 the active neuron on the surface of the map up to some
24 geometric distance defined by the neighbourhood
25 function) as though points closely connected by the
26 elastic net were being pulled towards the input by the
27 active neuron. This sequence of events is repeated
28 many times throughout the learning process as the
29 training data is fed to the system. At the start of
30 the learning process the elastic net is very flexible
31 due to large neighbourhoods and gain factor, but as
32 learning continues the net stiffens up as these
33 parameters become smaller. This process causes neurons
34 close together to form similar codebook values.

35
36 During this learning phase, the codebook vectors tend

1 to approximate various distributions of input vectors
2 with some sort of regularity and the resulting order
3 always reflects properties of the probability density
4 function $P(x)$ (ie the point density of the reference
5 vectors becomes proportional to $[P(x)]^{1/3}$). A similar
6 effect is found in biological neural systems where the
7 number of neurons within regions of the cortex
8 corresponding to different sensory modalities appear to
9 reflect the importance of the corresponding feature
10 set. The importance of a feature set is related
11 to the density of receptor cells connected to that
12 feature as would be expected. However, there also
13 appears to be a strong relationship between the number
14 of neurons representing a feature and the statistical
15 frequency of occurrence of that feature. The scale of
16 this relationship is often loosely referred to as
17 the magnification factor. While the reference vectors
18 are tending to describe the density function of inputs,
19 local interactions between neurons tend to preserve
20 continuity on the surface of the map. A combination of
21 these opposing forces causes the vector distribution to
22 approximate a smooth hyper-surface in the pattern space
23 with optimal orientation and form that best imitates
24 the overall structure of the input vector density.
25 This is done in such a way as to cause the map to
26 identify the dimensions of the feature space with
27 greatest variance which should be described in the map.
28 The initial ordering of the map occurs quite quickly
29 and is normally achieved within the first 10% of the
30 training phase, but convergence on optimal reference
31 vector values can take a considerable time. The
32 trained network provides a non-linear projection of the
33 probability density function $P(x)$ of the
34 high-dimensional input data x onto a 2-dimensional
35 surface (i.e. the surface of neurons).
36

1 Fig. 5 is a schematic representation of a single
2 modular map. At start-up time the Modular Map needs to
3 be configured with the correct parameter values for the
4 intended arrangement. All the 8-bit weight values are
5 loaded into the system at configuration time so that
6 the system can have either random weight values or
7 pre-trained values at start-up. The index of all
8 individual neurons, which consist of two 8-bit values
9 for the X and Y coordinates, are also selected at
10 configuration time. The flexibility offered by
11 allowing this parameter to be set is perhaps more
12 important for situations where several modules are
13 combined, but still offers the ability to create a
14 variety of network shapes for a stand alone situation.
15 For example, a module could be configured as a one or
16 two dimensional network. In addition to providing
17 parameters for individual neurons at configuration time
18 the parameters that apply to the whole network are also
19 required (i.e. the number of training steps, the gain
20 factor and neighbourhood start values). Intermediate
21 values for the gain factor and neighbourhood size are
22 then determined by the module itself during run time
23 using standard algorithms which utilise the current
24 training step and total number of training steps
25 parameters.

26
27 After configuration is complete, the Modular Map enters
28 its operational phase and data are input 16 Bits (i.e.
29 two input vector elements) at a time. The handshake
30 system controlling data input is designed in such a way
31 as to allow for situations where only a subset of the
32 maximum possible inputs is to be used. Due to
33 tradeoffs between data input rates and flexibility the
34 option to use only a subset of the number of possible
35 inputs is restricted to even numbers (i.e. 14, 12, 10
36 etc). However, if only say 15 inputs are required then

1 the 16th input element could be held constant for all
2 inputs so that it does not affect the formation of the
3 map during training. The main difference between the
4 two approaches to reducing input dimensionality is
5 that when the system is aware that inputs are not
6 present it does not make any attempt to use their
7 values to calculate the distance between the current
8 input and the codebook vectors within the network,
9 thereby reducing the workload on all neurons and
10 consequently reducing propagation time of the network.

11
12 After all inputs have been read by the Modular Map the
13 active neuron is determined and its X,Y coordinates are
14 output while the codebook vectors are being updated.
15 As the training process has the effect of creating a
16 topological map (such that neural activations across
17 the network have a meaningful order as though a feature
18 coordinate system were defined over the network) the
19 X,Y coordinates provide meaningful output. By feeding
20 inputs to the map after training has been completed it
21 is straightforward to derive an activation map which
22 could then be used to assign labels to the outputs from
23 the system.

24 25 Lateral Maps

26
27 As many difficult tasks require large numbers of
28 neurons the Modular Map has been designed to enable the
29 creation of networks with up to 65,536 neurons on a
30 single plane by allowing lateral expansion. Each
31 module consists of, for example, 256 neurons and
32 consequently this is the building block size for the
33 lateral expansion of networks. Each individual neuron
34 can be configured to be at any position on a
35 2-dimensional array measuring up to 256^2 but networks
36 should ideally be expanded in a regular manner so as to

1 create rectangular arrays. The individual neuron does
2 in fact have two separate
3 addresses; one is fixed and refers to the neuron's
4 location on the device and is only used locally; the
5 other, a virtual address, refers to the neuron's
6 location in the network and is set by the user at
7 configuration time. The virtual address is
8 accommodated by two 8-bit values denoting the X and Y
9 coordinates; it is these coordinates that are broadcast
10 when the active neuron on a module has been identified.

11
12 When modules are connected together in a lateral
13 configuration, each module receives the same input
14 vector. To simplify the data input phase it is
15 desirable that the data be made available only once for
16 the whole configuration of modules, as though only one
17 module were present. To facilitate this all modules in
18 the configuration are synchronised so that they act as
19 a single entity. The mechanism used to ensure this
20 synchronism is the data input handshake mechanism. By
21 arranging the input data bus for lateral configurations
22 to be inoperative until all modules are ready to accept
23 input, the modules will be synchronised. All the
24 modules perform the same functionality simultaneously,
25 so they can remain in synchronisation once it has been
26 established, but after every cycle new data is required
27 and the synchronisation will be reinforced.

28
29 All modules calculate the local 'winner' by using all
30 neurons on the module to simultaneously subtract one
31 from their calculated distance value until a neuron
32 reaches a value of zero. The first neuron to reach a
33 distance of zero is the one that initially had the
34 minimum distance value and is therefore the active
35 neuron for that module. The virtual coordinates of
36 this neuron are then output from the module, but

1 because all modules are synchronised, the first module
2 to attempt to output data is also the module containing
3 the 'global winner' (i.e. the active neuron for the
4 whole network). The index of the 'global winner' is
5 then passed to all modules in the configuration. When
6 a module receives this data it supplies it to all its
7 constituent neurons. Once a neuron receives this index
8 it is then able to determine if it is in the current
9 neighbourhood in exactly the same way as if it were
10 part of a stand alone module. Some additional logic
11 external to modules is required to ensure that only the
12 index which is output from the first module to respond
13 is forwarded to the modules in the configuration (see
14 Fig. 6). In Fig. 6, logic block A accepts as inputs
15 the data ready line from each module in the network.
16 The first module to set this line contains the "global
17 winner" for the network. When the logic receives this
18 signal it is passed to the device ready input which
19 forms part of the two line handshake used by all
20 modules in lateral expansion mode. When all modules
21 have responded to the effect that they are ready to
22 accept the coordinates of the active neuron the module
23 with these coordinates is requested by logic block A to
24 send the data. When modules are connected in this
25 lateral manner they work in synchronisation, and act as
26 though they were a single module which then allows them
27 to be further combined with other modules to form
28 larger networks.

29
30 Once a network has been created in this way it acts as
31 though it were a stand alone modular map and can be
32 used in conjunction with other modules to create a wide
33 range of network configurations. However, it should be
34 noted that as network size increases the number of
35 training steps also increases because the number of
36 training steps required is proportional to the network

1 size which suggests that maps are best kept to a
2 moderate size whenever possible.

3

4

5 Hierarchical Maps

6

7 The Modular Map system has been designed to allow
8 expansion by connecting maps together in different ways
9 to cater for changes in network size, and input vector
10 size, as well as providing the flexibility to enable
11 the creation of novel neural network configurations.
12 This modular approach offers a mechanism that maintains
13 an even workload among processing elements as systems
14 are scaled up, thereby providing an effective
15 parallelism of the Self Organising Map. To facilitate
16 expansion in order to cater for large input vectors,
17 modules are arranged in a hierarchical manner which
18 also appears plausible in terms of biological
19 systems where, for example, layers of neurons are
20 arranged in a hierarchical fashion in the primary
21 visual system with layers forming increasingly
22 complex representations the further up the hierarchy
23 they are situated.

24

25 Fig. 7 shows an example of a hierarchical network, with
26 four modules 10, 12, 14, 16 on the input layer I. The
27 output from each of the modules 12, 14, 16, 18 on the
28 input layer I is connected to the input of an output
29 module 18 on the output layer O. Each of the modules
30 10, 12, 14, 16, 18 has a 16 bit input data bus, and the
31 modules 10, 12, 14, 16 on the input layer I have 24
32 handshake lines connected as inputs to facilitate data
33 transfer between them, as will be described
34 hereinafter. The output module 18 has 12 handshake
35 lines connected as inputs, three handshake lines from
36 each of the modules 10, 12, 14, 16 in the input layer

1 I.

2

3 As each Modular Map is limited to a maximum of 16
4 inputs it is necessary to provide a mechanism which
5 will enable these maps to accept larger input
6 vectors so they may be applied to a wide range of
7 problem domains. Larger input vectors are accommodated
8 by connecting together a number of Modular Maps in
9 a hierarchical manner and partitioning the input data
10 across modules at the base of the hierarchy. Each
11 module in the hierarchy is able to accept up to 16
12 inputs, and outputs the X,Y coordinates of the active
13 neuron for any given input; consequently there is a
14 fan-in of eight modules to one which means that a
15 single layer in such a hierarchy will accept vectors
16 containing up to 128 inputs. By increasing the number
17 of layers in the hierarchy the number of inputs which
18 can be catered for also increases (i.e. Max Number of
19 inputs = $2 \cdot 8^n$ where n = number of layers in hierarchy).
20 From this simple equation it is apparent that very
21 large inputs can be catered for with very few layers in
22 the hierarchy.

23

24 By building hierarchical configurations of Modular Maps
25 to cater for large input vectors the system is in
26 effect parallelising the workload among many processing
27 elements. This approach was preferred over the
28 alternative of using more complex neurons which would
29 be able to accept larger input vectors. There
30 are many reasons for this, not least the problems
31 associated with implementation which, in the main,
32 dictate that hardware requirements increase with
33 increasing input vector sizes catered for.

34

35 Furthermore, as the input vector size increases, so too
36 does the workload on individual neurons which leads to

1 considerable increases in propagation delay through the
2 network. Hierarchical configurations keep the workload
3 on individual neurons almost constant, with an
4 increasing workload being met by an increase in neurons
5 used to do the work. It should be noted that there is
6 still an increase in propagation time with every layer
7 added to the hierarchy.

8
9 To facilitate hierarchical configurations of modular
10 maps it is necessary to ensure that communication
11 between modules is not going to form a bottleneck
12 which could adversely affect the operating speed of the
13 system. To circumvent this, a bus is provided to
14 connect the outputs from up to eight modules to the
15 input of a single module on the next layer of the
16 hierarchy (see Fig. 7). To avoid data collision and
17 provide sequence control, each Modular Map has 16 input
18 data lines plus three lines for each 16 bit input (two
19 vector elements), i.e. 24 handshake lines which
20 corresponds to a maximum of eight input devices.

21
22 Consequently, each module also has a three bit
23 handshake and 16 bit data output to facilitate the
24 interface scheme. One handshake line will be used to
25 advise the receiving module that the sender is present;
26 one line will be used to advise it that the sender is
27 ready to transmit data; and the third line will be used
28 to advise the sender that it should transmit the data.
29 After the handshake is complete the sender will then
30 place its data on the bus to be read by the receiver.
31 The simplicity of this approach negates the need for
32 additional interconnect hardware and thereby keeps to a
33 minimum the communication overhead. However, the
34 limiting factor with regard to these hierarchies and
35 their speed of operation is that each stage in the
36 hierarchy cannot be processed faster than the slowest

1 element at that level, but there are circumstances
2 under which the modules complete their classification
3 at differing rates and thereby affect operational
4 speed. For example, one module may be required to have
5 greater than the 256 neurons available to a single
6 Modular Map and would be made up of several maps
7 connected together in a lateral type of configuration
8 (as described above) which would slightly increase
9 the time required to determine its activations, or
10 perhaps a module has less than its maximum number of
11 inputs thereby reducing its time to determine
12 activations. It should also be noted that under normal
13 circumstances (i.e. when all modules are of equal
14 configurations) that the processing time at all layers
15 in the hierarchy will be the same as all modules are
16 carrying out equal amounts of work; this has the effect
17 of creating a pipelining effect such that throughput is
18 maintained constant even when propagation time through
19 the system is dependent on the number of layers in the
20 hierarchy.

21
22 As each Modular Map is capable of accepting a maximum
23 of 16 inputs and generates only a 2-dimensional output,
24 there is a dimensional compression ratio of 8:1
25 which offers a mechanism to fuse together many inputs
26 in a way that preserves the essence of the features
27 represented by those inputs with regard to the metric
28 being used.

29
30 An ordered network can be viewed in terms of regions of
31 activation surrounding the point positions of its
32 reference vectors, a technique sometimes referred
33 to as Voronoi sets. With this approach the whole of
34 the feature space is partitioned by hyper-planes
35 marking the boundaries of activation regions, which
36 contain all points from the input space that are closer

1 to the enclosed reference point than to any other point
2 in the network. These regions normally meet each other
3 in the same order as the topological arrangement
4 of neurons within the network. As with most techniques
5 applied to artificial neural networks, this approach is
6 only suitable for visualisation in two or three
7 dimensions, but can still be used to visualise what is
8 happening within hierarchical configurations of Modular
9 Maps. The series of graphs shown in Figs 8 to 10
10 emphasise some of the processes taking place in
11 hierarchical configurations. Although a 2-D data set
12 has been used for clarity, the processes identified
13 here are also applicable to higher dimensional data.

14
15 A Modular Map containing 64 neurons configured in a
16 square with neurons equally spaced within a 2-D plane
17 measuring 256^2 was trained on 2000 data points randomly
18 selected from two circular regions within the input
19 space of the same dimensions (see Fig. 8). The trained
20 network formed regions of activation as shown in the
21 Voronoi diagram of Fig. 9. From the map shown in Fig.
22 9 it is clear that the point positions of reference
23 vectors (shown as black dots) are much closer together
24 (i.e. have a higher concentration) around regions of
25 the input space with a high probability of containing
26 inputs. It is also apparent that, although a simple
27 distance metric (Manhattan distance) is being used by
28 neurons, the regions of activation can have some
29 interesting shapes. It should also be noted that the
30 formation of regions at the outskirts of the feature
31 space associated with the training data are often quite
32 large and suggest that further inputs to the trained
33 system considerably outwith the normal distribution of
34 the training data could lead to spurious neuron
35 activations. It was also observed that three neurons
36 of the trained network had no activations at all for

1 this data, the reference vector positions of these
2 three neurons (marked on the Voronoi diagram of Fig. 9
3 by *) fall between the two clusters shown and act as a
4 divider between the two classes.

5
6 As an approach to identifying the processes involved in
7 multidimensional hierarchies, the trained network
8 detailed in Fig. 9 was used to provide several inputs
9 to another network of the same configuration (except
10 the number of inputs) in a way that mimicked a four
11 into one hierarchy (i.e. four networks on the first
12 layer, one on the second). After the module at the
13 highest level in the hierarchy had been trained, it was
14 found that the regions of activation for the original
15 input space were as shown in Fig. 10. Comparison
16 between Figs 9 and 10 shows that the same regional
17 shapes have been maintained exactly, except that some
18 regions have been merged together, showing that
19 complicated non-linear regions can be generated in this
20 way without affecting the integrity of classification.
21 It can also be seen that the regions of activation
22 being merged together are normally situated where there
23 is a low probability of inputs so as to make more
24 efficient use of the resources available and provide
25 some form of compression. It should be noted that
26 there is an apparent anomaly because the activation
27 regions of the three neurons of the first network,
28 which are inactive after training, have not been merged
29 together, the reason being that this region of
30 inactivity is formed naturally between the two clusters
31 during training due to the 'elastic net' effect
32 outlined earlier and is consequently unaffected by the
33 merging of regions. This combining of regions has also
34 increased the number of inactive neurons to eight for
35 the second layer network. The processes highlighted
36 apply to higher dimensional data and suggest that such

1 hierarchical configurations not only provide a
2 mechanism for partitioning the workload of large input
3 vectors, but can also provide a basis for data fusion
4 of a range of data types, from different sources and
5 input at different stages in the hierarchy.
6

7 When modules are connected together in a hierarchical
8 manner there is still the opportunity to partition
9 input data in various ways. The most obvious approach
10 is to simply split the original high dimensional input
11 data into vectors of 16 inputs or less, i.e. given the
12 original feature space \mathbb{R}^n , n is partitioned into groups
13 of 16 or less. When data is partitioned in this way,
14 each module forms a map of its respective input domain,
15 there is no overlap of maps, and a module has no
16 interaction with other modules on its level in the
17 hierarchy. However, it is also realistic to consider
18 an approach where inputs to the system would span more
19 than one module, thereby enabling some data overlap
20 between modules. An approach of this nature can assist
21 modules in their classification by providing them with
22 some sort of context for the inputs; it is also a
23 mechanism which allows the feature space to be viewed
24 from a range of perspectives with the similarity
25 between views being determined by the extent of the
26 data overlap. Simulations have also shown that an
27 overlap of inputs (i.e. feeding some inputs to two or
28 more separate modules) can lead to an improved mapping
29 and classification.
30

31 A similar approach to partitioning could also be taken
32 to give better representation to the range of values in
33 any dimension, i.e. \mathbb{R} could be partitioned.
34 Partitioning a single dimension of the feature space
35 across several inputs should not normally be required,
36 but if the reduced range of 256 which is available to

1 the Modular Map should prove to be too restrictive for
2 an application, then the flexibility of the Modular Map
3 is able to support such a partitioning approach. The
4 range of values supported by the Modular Map inputs
5 should be sufficient to capture the essence of any
6 single dimension of the feature space, but
7 pre-processing is normally required to get the best out
8 of the system.

9
10 Partitioning \mathcal{R} is not as simple as partitioning n , and
11 would require a little more pre-processing of input
12 data, but the approach could not be said to be overly
13 complex. However, when partitioning \mathcal{R} , only one of the
14 inputs used to represent each of the feature space
15 dimensions will contain input stimuli for each input
16 pattern presented to the system. Consequently, it is
17 necessary to have a suitable mechanism to cater for
18 this eventuality, and the possible solutions are to
19 either set the system input to the min or max value
20 depending on which side of the domain of this input the
21 actual input stimuli is on, or do not use an input at
22 all if it does not contain active input stimuli.

23
24 The design of the Modular Map is of such flexibility
25 that inputs could be partitioned across the network
26 system in some interesting ways, e.g. inputs could be
27 taken directly to any level in the hierarchy.
28 Similarly, outputs can also be taken from any module in
29 the hierarchy, which may be useful for merging or
30 extracting different information types. There is no
31 compulsion to maintain symmetry within a hierarchy
32 which could lead to some novel configurations, and
33 consequently separate configurations could be used for
34 specific functionality and combined with other modules
35 and inputs to form systems with increasing complexity
36 of functionality. It is also possible to introduce

1 feedback into Modular Map systems which may enable the
2 creation of some interesting modular architectures and
3 expand possible functionality.

4

5

6 Neural Pathways and Hybrid Networks

7

8 Various types of sensory modalities such as light,
9 sound and smell are mapped to different parts of the
10 brain. Within each of these modalities specific
11 stimuli, e.g. lines or corners in the visual system,
12 act selectively on specific populations of neurons
13 situated in different regions of the cortex. The
14 number of neurons within these regions reflect the
15 importance of the corresponding feature set. The
16 importance of a feature set is related to the density
17 of receptor cells connected to that feature. However,
18 there is also a strong relationship between the number
19 of neurons representing a feature and the statistical
20 frequency of occurrence of that feature. The scale of
21 this relationship is often loosely referred to as the
22 magnification factor.

23

24 While the neocortex contains a great many neurons,
25 somewhere in the region of 10^9 , it only contains two
26 broad categories of neuron; smooth neurons and spiny
27 neurons. All the neurons with spines (pyramidal cells
28 and spiny stellates) are excitatory and all smooth
29 neurons (smooth stellates) are inhibitory. The signals
30 presented to neurons are also limited to two types of
31 electrical message. The mechanisms by which these
32 signals are generated are similar throughout the brain
33 and the signals themselves cannot be endowed with
34 special properties because they are stereotyped and
35 much the same in all neurons. It seems that with such a
36 limited range of components with stereotyped signals

1 that the connections will have an important bearing on
2 the capabilities of the brain.

3

4 It may be possible to facilitate dynamically changing
5 context dependent pathways within Modular Map systems
6 by utilising feedback and the concepts of excitory and
7 inhibitory neurons as found in nature. This prospect
8 exists because the interface of a Modular Map allows
9 for the processing of only part of the input vector,
10 and supports the possibility of a module being
11 disabled. The logic for such inhibitory systems would
12 be external to the modules themselves, but could
13 greatly increase the flexibility of the system. Such
14 inhibition could be utilised in several ways to
15 facilitate different functionality, e.g. either some
16 inputs or the output of a module could be inhibited.
17 If insufficient inputs were available a module or
18 indeed a whole neural pathway could be disabled for a
19 single iteration, or if the output of a module were to
20 be within a specific range then parts of the system
21 could be inhibited. Clearly, the concept of an
22 excitory neuron would be the inverse of the above with
23 parts of the system only being active under specific
24 circumstances.

25

26 When implementing ANNs in hardware difficulties are
27 encountered as network size increases. The underlying
28 reasons for this are silicon area, pin out
29 considerations and inter-processor communications. By
30 utilising a modular approach towards implementation,
31 the inherent partitioning strategy overcomes the usual
32 limitations on scaleability. Only a small number of
33 neurons are required for a single module and separate
34 modules are implemented on separate devices.

35

36 The Modular Map design is fully digital and uses a fine

1 grain implementation approach, i.e. each neuron is
2 implemented as a separate processing element. Each of
3 these processing elements is effectively a simple
4 Reduced Instruction Set Computer (RISC) with limited
5 capabilities, but sufficient to perform the
6 functionality of a neuron. The simplicity of these
7 neurons has been promoted by applying modifications to
8 Kohonen's original algorithm. These modifications have
9 also helped to minimise the hardware resources required
10 to implement the Modular Map design.

11

12 Background

13

14 Essentially the Self-Organising Map (SOM) consists of a
15 two dimensional array of neurons connected together by
16 strong lateral connections. Each neuron has its own
17 reference vector which input vectors are measured
18 against. When an input vector is presented to the
19 network, it is passed to all neurons constituting the
20 network. All neurons then proceed to measure the
21 similarity between the current input vector and their
22 local reference vectors. This similarity is assessed
23 by calculating the distance between the input vector
24 and the reference vector, generally using the Euclidean
25 distance metric. In the Modular Map implementation
26 Euclidean distance is replaced by Manhattan distance
27 because Manhattan distance can be determined using only
28 an adder/subtractor unit whereas calculations of
29 Euclidean distances require determination of the
30 squares of differences involved and would therefore
31 require a multiplier unit which would use considerably
32 greater hardware resources.

33

34 There are a range of techniques that could be utilised
35 to perform the multiplication operations required to
36 calculate Euclidean distance. These include multiple

1 addition operations, which would introduce unacceptable
2 time delays, or traditional multiplier units such as a
3 Braun's multiplier, but compared to an adder/subtractor
4 unit the resource requirements would be significantly
5 increased. There would also be an increase in the time
6 required to obtain the result of a multiplication
7 operation compared to the addition/subtraction required
8 to calculate Manhattan distance. Furthermore, when
9 using multiplication, the number of bits in the result
10 is equal to the number of bits in the multiplicand plus
11 the number of bits in the multiplier, which would
12 produce a 16 bit result for an 8 bit by 8 bit
13 multiplication and would therefore require at least a
14 16 bit adder to calculate the sum of distances. This
15 requirement would further increase the resource
16 requirements for calculating Euclidean distance and,
17 consequently, further increases the advantages of using
18 the Manhattan distance metric.

19
20 Once all neurons in the network have determined their
21 respective distances they communicate via strong
22 lateral connections with each other to determine which
23 amongst them has the minimum distance between its
24 reference vector and the current input. The Modular Map
25 implementation maintains strong local connections, but
26 determination of the winner is achieved without the
27 communications overhead suggested by Kohonen's original
28 algorithm. All neurons constituting the network are
29 used in the calculations to determine the active neuron
30 and the workload is spread among the network as a
31 result.

32
33 During the training phase of operation all neurons in
34 the immediate vicinity of the active neuron update
35 their reference vectors to bring them closer to the
36 current input. The size of this neighbourhood changes

1 throughout the training phase, initially being very
2 large and finally being restricted to the active neuron
3 itself. The shape of neighbourhood can take on many
4 forms, the two most popular being a square step
5 function and a gaussian type neighbourhood. The
6 Modular Map approach again utilises Manhattan distance
7 to measure the neighbourhood, which results in a square
8 neighbourhood, but it is rotated through 45 degrees so
9 that it appears to be a diamond shape (Fig. 3). This
10 further assists the implementation because an
11 adder/subtractor unit is still all that is required at
12 this stage. However, additional hardware is required
13 to update reference vector values because reference
14 vectors are only updated by a proportion of the
15 distance between the input and reference vectors. The
16 proportionality of the update applied is determined by
17 what is normally referred to as the gain factor $\alpha(t)$
18 which Kohonen specifies as a decreasing monotonic
19 function. Consequently, a mechanism is required that
20 will enable multiplication of distances by a suitable
21 range of fractional values. This is achieved by
22 restricting $\alpha(t)$ to negative powers of two. By
23 restricting $\alpha(t)$ in this way it is possible to perform
24 the required multiplication by using only an arithmetic
25 shifter, which is considerably less expensive in terms
26 of hardware resources than a full multiplier unit.

27

28

29 The Neuron

30

31 The Modular Map approach has resulted in a simple
32 Reduced Instruction Set Computer (RISC) type
33 architecture for neurons. The key elements of the
34 neuron design which are shown in Fig. 11 are an
35 adder/subtractor unit (ALU) 50, a shifter mechanism 52,
36 a set of registers and control logic 54. The ALU 50 is

1 the main computational component and by utilising an
2 arithmetic shifter mechanism 52 to perform all
3 multiplication functions, the ALU 50 requirements have
4 been kept to a minimum.

5
6 All registers in a neuron are individually addressable
7 as 8 or 12 bit registers although individual bits are
8 not directly accessible. Instructions are received by
9 the neuron from the module controller and the local
10 control logic interprets these instructions and
11 coordinates the operations of the individual neuron.
12 This task is kept simple by maintaining a simple series
13 of instructions that only number thirteen in total.

14
15 The adder/subtractor unit 50 is clearly the main
16 computational element within a neuron. The system
17 needs to be able to perform both 8 bit and 12 bit
18 arithmetic, with 8 bit arithmetic being the most
19 frequent. A single 4 bit adder/subtractor unit could
20 be utilised to do both the 8 bit and 12 bit arithmetic,
21 or an 8 bit unit could be used. However, there will be
22 considerably different execution times for different
23 sizes of data if a 12 bit adder/subtractor unit is not
24 used (e.g. if an 8 bit unit is used it will take
25 approximately twice as long to perform 12 bit
26 arithmetic as it would 8 bit arithmetic because two
27 passes through the adder/subtractor would be required).
28 In order to avoid variable execution times for the
29 different calculations to be performed a 12 bit
30 adder/subtractor unit is preferable.

31
32 A 12 bit adder/subtractor unit utilising a Carry
33 Lookahead Adder (CLA) would require approximately 160
34 logic gates, and would have a propagation delay equal
35 to the delay of 10 logic gates. The ALU 50 also has
36 two flags and two registers directly associated with

1 it. The two flags associated with the ALU 50 are a
2 zero flag, which is set when the result of an
3 arithmetic operation is zero, and a negative flag,
4 which is set when the result is negative.

5
6 The registers associated with the ALU 50 are both 12
7 bit; a first register 56 is situated at the ALU output;
8 a second register 58 is situated at one of the ALU
9 inputs. The first register 56 at the output from the
10 ALU 50 is used to buffer data until it is ready to be
11 stored. Only a single 12 bit register 58 is required
12 at the input to the ALU 50 as part of an approach that
13 allows the length of instructions to be kept to a
14 minimum. The design is a register-memory architecture,
15 and arithmetic operations are allowed directly on
16 register values but the instruction length used for the
17 neuron is too small to include an operation and the
18 addresses of two operands in a single instruction.
19 Thus, the second register 58 at one of the ALU inputs
20 is used so that the first datum can be placed there for
21 use in any following arithmetic operations. The
22 address of the next operand can be provided with the
23 operator code and, consequently, the second datum can
24 be accessed directly from memory.

25
26 The arithmetic shifter mechanism 52 is only required
27 during the update phase of operation to multiply the
28 difference between input and weight elements by the
29 gain factor value $\alpha(t)$. The gain factor $\alpha(t)$ is
30 advantageously restricted to four values (i.e. 0.5,
31 0.25, 0.125 and 0.0625). Consequently, the shifter
32 mechanism 52 is required to shift right by 0, 1, 2, 3
33 and 4 bits to perform the required multiplication. The
34 arithmetic shifter 52 can typically be implemented
35 using flip flops which is a considerable improvement on
36 the alternative of a full multiplier unit which would

1 require substantially more resources to implement.

2

3 It should be noted that, for the bit shift approach to
4 work correctly, weight values are required to have as
5 many additional bits as there are bit shift operations
6 (i.e. given that a weight value is 8 bits, when 4 bit
7 shifts are allowed, 12 bits need to be used for the
8 weight value). The additional bits store the
9 fractional part of weight values and are only used
10 during the update operation to ensure convergence is
11 possible; there is no requirement to use this
12 fractional part of weight values while determining
13 Manhattan distance.

14

15 For simplicity with flexibility the arithmetic shifter
16 52 is positioned in the data stream between the output
17 of the ALU 50 and its input register 58, but is only
18 active when the gain value is greater than zero. This
19 approach was regarded as a suitable approach to
20 limiting the number of separate instructions because
21 the gain factor values are supplied by the system
22 controller at the start of the update phase of
23 operations and can be reset to zero at the end of this
24 operational phase.

25

26 The data registers of these RISC neurons require
27 substantial resources and must hold 280 bits of data.
28 The registers must be readily accessible by the neuron,
29 especially the reference vector values which are
30 accessed frequently. In order for the system to
31 operate effectively access to weight values is required
32 either 8 or 12 bits at a time for each neuron,
33 depending on the phase of operation. This requirement
34 necessitates on-chip memory because there are a total
35 of 64 neurons attempting to access their respective
36 weight values simultaneously. This results in a

51

1 minimum requirement of 512 bits rising to 768 bits
2 (during the update phase) that need to be accessed
3 simultaneously. Clearly, this would not be possible if
4 the weight values were stored off chip because a single
5 device would not have enough I/O pins to support this
6 in addition to other I/O functions required of a
7 Modular Map. There are ways of maximising data access
8 with limited pin outs but, a bottleneck situation could
9 not be entirely avoided if memory were off chip.

10

11 The registers are used to hold reference vector values
12 (16*12 bits), the current distance value (12 bits), the
13 virtual X and Y coordinates (2*8 bits), the
14 neighbourhood size (8 bits) and the gain value $\alpha(t)$ (3
15 bits) for each neuron. There are also input and output
16 registers (2*8bits), registers for the ALU (2*12), a
17 register for the neuron ID (8 bit) and a one bit
18 register for maintaining an update flag. Of these
19 registers all can be directly addressed except for the
20 output register and update flag, although the neuron ID
21 is fixed throughout the training and operational
22 phases, and like the input register is a read only
23 register as far as the neuron is concerned.

24

25 At start up time all registers except the neuron ID are
26 set to zero values before parameter values are provided
27 by an I/O controller. At this stage the initial weight
28 values are provided by the controller to allow the
29 system to start from either random weight values or
30 values previously determined by training a network.
31 While 12 bit registers are used to hold the weight
32 values, only 8 bits are used for determining a neuron's
33 distance from an input, and only these 8 bits are
34 supplied by the controller at start up; the remaining 4
35 bits represent the fractional part of the weight value,
36 are initially set to zero, and are only used during

1 weight updates.

2

3 The neighbourhood size is also supplied by the
4 controller at start up but, like the gain factor $\alpha(t)$,
5 it is a global variable that changes throughout the
6 training process requiring new values to be effected by
7 the controller at appropriate times throughout
8 training. The virtual coordinates are also provided by
9 the controller at start up time, but are fixed
10 throughout the training and operational phases of the
11 system and provide the neuron with a location from
12 which to determine if it is within the current
13 neighbourhood. Because virtual addresses are used for
14 neurons, any neuron can be configured to be anywhere
15 within a 256^2 array which provides great flexibility
16 when networks are combined to form systems using many
17 modules. It is advantageous for the virtual addresses
18 used in a network to maximise the virtual address space
19 (i.e. use the full range of possible addresses in both
20 the X and Y dimensions). For example, if a 64 neuron
21 module is used, the virtual addresses of neurons along
22 the Y axis should be 0,0 0,36 0,72 etc. In this way
23 the outputs from a module will utilise the maximum
24 range of possible values, which in this instance will
25 be between 0 and 252. Simulations found that
26 classification results were poor when this practice was
27 not adopted.

28

29 It should also be noted that, because there is a
30 requirement to use mixed sizes of data, an update flag
31 is used as a switch mechanism for the data type to be
32 used. This mechanism was found to be necessary because
33 when 8 bit values and 12 bit values are being used
34 there are differing requirements at different phases of
35 operation. During the normal operational phase only 8
36 bit values are necessary but they are required to be

1 the least significant 8 bits, e.g. when calculating
2 Manhattan distance. However, during the update phase
3 of operation both 8 bit and 12 bit values are used.
4 During this update phase all the 8 bit values are
5 required to be the most significant 8 bits and when
6 applying changes to reference vectors the full 12 bit
7 value is required. By using a simple flag as a switch
8 the need for duplication of instructions is avoided so
9 that operations on 8 and 12 bit values can be executed
10 using the same instruction set.

11
12 The control logic within a neuron is kept simple and is
13 predominantly just a switching mechanism. All
14 instructions are the same size, i.e. 8 bits, but there
15 are only thirteen distinct instructions in total.
16 While an 8 bit instruction set would in theory support
17 256 separate instructions, one of the aims of the
18 neuron design has been to use a reduced instruction
19 set. In addition, separate registers within a neuron
20 need to be addressable to facilitate all the operations
21 required of them and, where an instruction needs to
22 refer to a particular register address, that address
23 effectively forms part of the instruction.

24
25 The instruction length has been set at 8 bits because
26 the data bus is only 8 bits wide which sets the upper
27 limit for a single cycle instruction read. There is
28 also a requirement to address locations of operands for
29 six of the instructions which necessitates the
30 incorporation of up to 25 separate addresses into these
31 instructions and will require 5 bits for the address of
32 the operand alone. However, the total instruction
33 length can still be maintained at 8 bits because
34 instructions that do not require operand addresses can
35 use some of these bits as part of their instruction
36 and, consequently, there is room for expansion of the

1 instruction set within the instruction space.

2

3 All instructions for neuron operations are 8 bits in
4 length and are received from the controller. The first
5 input to a neuron is always an instruction, normally
6 the reset instruction to zero all registers. The
7 instruction set is as follows:

8

9 RDI: (Read Input) will read the next datum from its
10 input and write to the specified register address.
11 This instruction will not affect arithmetic flags.

12

13 WRO: (Write arithmetic Output) will move the current
14 data held at the output register 56 of the ALU to the
15 specified register address. This instruction will
16 overwrite any existing data in the target register and
17 will not affect the systems arithmetic flags.

18

19 ADD: Add the contents of the specified register
20 address to that already held at the ALU input. This
21 instruction will affect arithmetic flags and, when the
22 update register is zero all 8 bit values will be used
23 as the least significant 8 bits of the possible 12, and
24 only the most significant 8 bits of weight vectors will
25 be used (albeit as the least significant 8 bits for the
26 ALU) when the register address specified is that of a
27 weight whereas, when the update register is set to one,
28 all 8 bit values will be set as the most significant
29 bits and all 12 bits of weight vectors will be used.

30

31 SUB: Subtract the value already loaded at the ALU
32 input from that at the specified register address.
33 This instruction will affect arithmetic flags and will
34 treat data according to the current value of the update
35 register as detailed for the add command.

36

1 BRN: (Branch if Negative) will test the negative flag
2 and will carry out the next instruction if it is set,
3 or the next instruction but one if it is not.
4
5 BRZ: (Branch if Zero) will test the zero flag and will
6 carry out the next instruction if it is set. If the
7 flag is zero the next but one instruction will be
8 executed.
9
10 BRU: (Branch if Update) will test the update flag and
11 will carry out the next instruction if it is set, or
12 the next instruction but one if it is not.
13
14 OUT: Output from the neuron the value at the specified
15 register address. This instruction does not affect the
16 arithmetic flags.
17
18 MOV: Set the ALU input register to the value held in
19 the specified address. This instruction will not
20 affect the arithmetic flags.
21
22 SUP: Set the update register. This instruction does
23 not affect the arithmetic flags.
24
25 RUP: Reset the update register. This instruction does
26 not affect the arithmetic flags.
27
28 NOP: (No Operation) This instruction takes no action
29 for one instruction cycle.
30
31 MRS: Master reset will reset all registers and flags
32 within a neuron to zero.
33
34
35 **The Module Controller**
36

1 Fig. 12 shows a schematic representation of a module
2 controller for controlling the operation of a number of
3 RISC neurons, one of which is shown in Fig. 11. The
4 Module Controller is required to handle all device
5 input and output in addition to issuing instructions to
6 neurons within a module and synchronising their
7 operations. To facilitate these operations the
8 controller system comprises the I/O ports 60, 62; a
9 programmable read-only-memory (PROM) 64 containing
10 instructions for the controller system and subroutines
11 for the neural array; an address map 66 for conversion
12 between real and virtual neuron addresses; an input
13 buffer 68 to hold incoming data; and a number of
14 handshake mechanisms (see Fig. 12).

15
16 The controller handles all input for a module which
17 includes start-up data during system configuration, the
18 input vectors 16 bits (two vector elements) at a time
19 during normal operation, and also the index of the
20 active neuron when configured in lateral expansion
21 mode. Outputs from a module are also handled
22 exclusively by the controller. The outputs are limited
23 to a 16 bit output representing Cartesian coordinates
24 of the active neuron during operation and parameters of
25 trained neurons such as their weight vectors after
26 training operations have been completed. To enable the
27 above data transfers a bi-directional data bus is
28 required between the controller and the neural array
29 such that the controller can address either individual
30 neurons or all neurons simultaneously; there is no
31 requirement to allow other groups of neurons to be
32 addressed but the bus must also carry data from
33 individual neurons to the controller.

34
35 While Modular Map systems are intended to allow modules
36 to operate asynchronously from each other, except when

1 in lateral expansion mode it is necessary to
2 synchronise data communication in order to simplify the
3 mechanism required. When two modules have a data
4 connection linking them together a handshake mechanism
5 is used to synchronise data transfer from the module
6 transmitting the data (the sender) to the module
7 receiving the data (the receiver). The handshake is
8 implemented by the module controllers of the sender and
9 receiver modules, only requires three handshake lines
10 and can be viewed as a state machine with only three
11 possible states:

12

- 13 1) Wait (Not ready for input)
- 14 2) No Device (No input stream for this position)
- 15 3) Data Ready (Transfer data)

16

17 The handshake system is shown as a simple state diagram
18 in Fig. 13. With reference to Fig. 13, the wait state
19 70 occurs when either the sender or receiver (or both)
20 are not ready for data transfer. The no device state
21 72 is used to account for situations where inputs are
22 not present so that reduced input vector sizes can be
23 utilised. This mechanism could also be used to
24 facilitate some fault tolerance when input streams are
25 out of action so that the system did not come to a
26 halt. The data ready state 74 occurs when both the
27 sender and the receiver are ready to transfer data and,
28 consequently, data transfer follows immediately this
29 state is entered. This handshake system makes it
30 possible for a module to read input data in any
31 sequence. When a data source is temporarily
32 unavailable the delay can be minimised by processing
33 all other input vector elements while waiting for that
34 datum to become available. Individual neurons could
35 also be instructed to process inputs in a different
36 order but, as the controller buffers input data there

1 is no necessity for neurons to process data in the same
2 order it is received. The three possible conditions of
3 this data transfer state machine are determined by two
4 outputs from the sender module and one output from the
5 receiving module. The three line handshake mechanism
6 allows the transfer of data direct to each other
7 wherein no third party device is required, and data
8 communication is maintained as point to point.

9
10 Similarly, data is also output 16 bits at a time, but
11 as there are only two 8 bit values output by the
12 system, only a single data output cycle is required,
13 with the three line handshake mechanism used to
14 synchronise the transfer of data, three handshake
15 connections are also required at the output of a
16 module. However, the inputs are intended to be
17 received from up to eight separate sources, each one
18 requiring three handshake connections thereby giving a
19 total of 24 handshake connections for the input data.
20 This mechanism will require 24 pins on the device but,
21 internal multiplexing will enable the controller to use
22 a single three line handshake mechanism internally to
23 cater for all inputs.

24
25 To facilitate reading the coordinates for lateral
26 expansion mode, a two line handshake system is used.
27 The mechanism is similar to the three line handshake
28 system, except the 'device not present' state is
29 unnecessary and has therefore been omitted.

30
31 The module controller is also required to manage the
32 operation of neurons on its module. To facilitate such
33 control there is a programmable read-only memory (PROM)
34 64 which holds subroutines of code for the neural array
35 in addition to the instructions it holds for the
36 controller. The program is read from the PROM and

1 passed to the neural array a single instruction at a
2 time. Each instruction is executed immediately when
3 received by individual neurons. When issuing these
4 instructions the controller also forwards incoming data
5 and processes outgoing data. There are four main
6 routines required to support full system functionality
7 plus routines for setting up the system at start up
8 time and outputting reference vector values etc. at
9 shutdown. The start up and shutdown routines are very
10 simple and only require data to be written to and read
11 from registers using the RDI and OUT commands. The
12 four main routines are required to enable the
13 calculation of Manhattan distance (calcdist); find the
14 active neuron (findactive); determine which neurons are
15 in the current neighbourhood (nbhood); and update
16 reference vectors (update). Each of these procedures
17 will be detailed in turn.

18
19 The most frequently used routine (calcdist) is required
20 to calculate the Manhattan distance for the current
21 input. When an input vector is presented to the system
22 it is broadcast to all neurons an element at a time,
23 (i.e. each 8 bit value) by the controller. As neurons
24 receive this data they calculate the distance between
25 each input value and its corresponding weight value,
26 adding the results to the distance register. The
27 controller reads the routine from the program ROM,
28 forwards it to the neural array and forwards the
29 incoming data at the appropriate time. This subroutine
30 is required for each vector element and will be as
31 follows:

32
33 MOV (W_i) /*Move weight (W_i) to the ALU input
34 register.*/
35 SUB (X_i) /*Subtract the value at the ALU register from
36 the next input.*/
37

60

```
1  MOV (Ri)  /*Move the result (Ri) to the ALU input
2           register.*/
3  BRN       /*If the result was negative*/
4  SUB dist  /*distance = distance - Ri*/
5  ADD dist  /*Else distance = distance + Ri*/
6  WRO dist  /*Write the new distance to its register.*/
7
8  Once all inputs have been processed and neurons have
9  calculated their respective Manhattan distances the
10 active neuron needs to be identified. As the active
11 neuron is simply the neuron with minimum distance and
12 all neurons have the ability to make these calculations
13 the workload can be spread across the network. This
14 approach can be implemented by all neurons
15 simultaneously subtracting one from their current
16 distance value repeatedly until a neuron reaches a zero
17 distance value, at which time it would poll the
18 controller to notify it that it was the active neuron.
19 Throughout this process the value to be subtracted from
20 the distance is supplied to the neural array by the
21 controller. On the first iteration this will be zero
22 to check if any neuron has a match with the current
23 input vector (i.e. distance is already zero) thereafter
24 the value forwarded will be one. The subroutine
25 findactive defines this process as follows:
26
27
28 MOV input /*Move the input to the ALU input register.*/
29 SUB dist  /*Subtract the next input from the current
30           distance value.*/
31 BRZ       /*If result is zero.*/
32 OUT ID    /*output the neuron ID.*/
33 NOP       /*Else do nothing.*/
34
35 On receiving an acknowledge signal from one of the
36 neurons in the network, by way of its ID, the
```

1 controller would output the virtual coordinates of the
2 active neuron. The controller uses a map (or lookup
3 table) of these coordinates which are 16 bits so that
4 neurons can pass only their local ID (8 bits) to the
5 controller. It is important that the controller
6 outputs the virtual coordinates of the active neuron
7 immediately they become available because when
8 hierarchical systems are used the output is required to
9 be available as soon as possible for the next layer to
10 begin processing the data, and when modules are
11 configured laterally it is not possible to know the
12 coordinates of the active neuron until they have been
13 supplied to the input port of the module.

14
15 When modules are connected together in a lateral
16 manner, each module is required to output details of
17 the active neuron for that device before reference
18 vectors are updated because the active neuron for the
19 whole network may not be the same as the active neuron
20 for that particular module. When connected together in
21 this way, modules are synchronised and the first module
22 to respond is the one containing the active neuron for
23 the whole network. Only the first module to respond
24 will have its output forwarded to the inputs of all the
25 modules constituting the network. Consequently, no
26 module is able to proceed with updating reference
27 vectors until the coordinates of the active neuron have
28 been supplied via the input of the device because the
29 information is not known until that time. When a
30 module is in 'lateral mode' the two line handshake
31 system is activated and after the coordinates of the
32 active neuron have been supplied the output is reset
33 and the coordinates broadcast to the neurons on that
34 module.

35
36 When coordinates of the active neuron are broadcast,

```
1  all neurons in the network determine if they are in the
2  current neighbourhood by calculating the Manhattan
3  distance between the active neurons virtual address and
4  their own. If the result is less than or equal to the
5  current neighbourhood value, the neuron will set its
6  update flag so that it can update its reference vector
7  at the next operational phase. The routine for this
8  process (nbhood) is as follows:
9
10
11  MOV Xcoord      /*Move the virtual X coordinate to the
12                  ALU input register.*/
13  SUB input       /*Subtract the next input (X coord) from
14                  value at ALU.*/
15  WRO dist        /*Write the result to the distance
16                  register.*/
17  MOV Ycoord      /*Move the virtual Y coordinate the
18                  ALU.*/
19  SUB input       /*Subtract the next input (Y coord) from
20                  value at ALU.*/
21  MOV dist        /*Move the value in distance register to
22                  ALU.*/
23  ADD result      /*Add the result of the previous
24                  arithmetic to the value at ALU input.*/
25  MOV result      /*Move the result of the previous
26                  arithmetic to the ALU input.*/
27  SUB input       /*Subtract the next input (neighbourhood
28                  val) from value at ALU.*/
29  BRN             /*If the result is negative.*/
30  SUP             /*Set the update flag.*/
31  BRZ             /*If the result is zero.*/
32  SUP             /*Set the update flag.*/
33  NOP            /*Else do nothing*/
34
35  All neurons in the current neighbourhood then go on to
36  update their weight values. To achieve this they also
```

1 have to recalculate the difference between input and
2 weight elements, which is inefficient computationally
3 as these values have already been calculated in the
4 process of determining Manhattan distance. However,
5 the alternative would require these intermediate values
6 to be stored by each neuron, thereby necessitating an
7 additional 16 bytes of memory per neuron. To minimise
8 the use of hardware resources these intermediate values
9 are recalculated during the update phase. To
10 facilitate this the module controller stores the
11 current input vector and is able to forward vector
12 elements to the neural array as they are required. The
13 update procedure is then executed for each vector
14 element as follows:

```
15
16 RDI gain /*Read next input and place it in the gain
17          register.*/
18 MOV Wi /*Move weight value (Wi) to ALU input.*/
19 SUB input /*Subtract the input from value at ALU*/
20 MOV result /*Move the result to the ALU. */
21 ADD Wi /*Add weight value (Wi) to ALU input.*/
22 BRU /*If the update flag is set.*/
23 WRO Wi /*Write the result back to the weight
24          register.*/
25 NOP /*Else do nothing.*/
```

26
27 After all neurons in the current neighbourhood have
28 updated their reference vectors the module controller
29 reads in the next input vector and the process is
30 repeated. The process will then continue until the
31 module has completed the requested number of training
32 steps or an interrupt is received from the master
33 controller. The term 'master controller' is used to
34 refer to any external computer system that is used to
35 configure Modular Maps. The master controller is not
36 required during normal operation as Modular Maps

1 operate autonomously but is required to supply the
2 operating parameters and reference vector values at
3 start up time, set the mode of operation and collect
4 the network parameters after training is completed.
5 Consequently, the module controller receives
6 instructions from the master controller at these times.
7 To enable this, modules have a three bit instruction
8 interface exclusively for receiving input from the
9 master controller. The instructions received are very
10 basic and the total master controller instruction set
11 only comprises six instructions which are as follows:
12
13

14 RESET: This is the master reset instruction and is
15 used to clear all registers etc. in the controller and
16 neural array
17

18 LOAD: Instructs the controller to load in all the
19 setup data for the neural array including details
20 of the gain factor and neighbourhood parameters. The
21 number of data items to be loaded is constant for all
22 configurations and data are always read in the same
23 sequence. To enable data to be read by the controller
24 the normal data input port is used with a two line
25 handshake (the same one used for lateral mode), which
26 is identical to the three line handshake described
27 earlier, except that the device present line is not
28 used.
29

30 UNLOAD: Instructs the controller to output network
31 parameters from a trained network. As with the LOAD
32 instruction the same data items are always output in
33 the same sequence. The data are output from the
34 modules data output port.
35

36 NORMAL: This input instructs the controller to run in

1 normal operational mode

2

3 LATERAL: This instructs the controller to run in
4 lateral expansion mode. It is necessary to have this
5 mode separate to normal operation because the module is
6 required to read in coordinates of the active neuron
7 before updating the neural arrays reference vectors and
8 reset the output when these coordinates are received.

9

10 STOP: This is effectively an interrupt to advise
11 the controller to cease its current operation.

12

13

14 The Module

15

16 An individual neuron is of little use on its own, the
17 underlying philosophy of neural networks dictates that
18 they are required in groups to enable parallel
19 processing and perform the levels of computation
20 necessary to solve computationally difficult problems.
21 The minimum number of neurons that constitute a useful
22 group size is debatable and is led more by the problem
23 to be addressed (i.e. the application) than by any
24 other parameters. It is desirable that the number of
25 neurons on a single module be small enough to enable
26 implementation on a single device. Another
27 consideration was motivated by the fact that Modular
28 Maps are effectively building blocks that are intended
29 to be combined to form larger systems. As these
30 factors are interrelated and can affect some network
31 parameters such as neighbourhood size, it was decided
32 that the number of neurons would be a power of 2 and
33 the network size which best suited these requirements
34 was 256 neurons per module.

35

36 As the Modular Map design is intended for digital

1 hardware there are a range of technologies available
2 that could be used, e.g. full custom very large scale
3 integration (VLSI), semi-custom VLSI, application
4 specific integrated circuit (ASIC) or Field
5 Programmable Gate Arrays (FPGA). A 256 neuron Modular
6 Map constitutes a small neural network and the
7 simplicity of the RISC neuron design leads to reduced
8 hardware requirements compared to the traditional SOM
9 neuron.

10

11 The Modular Map design maximises the potential for
12 scalability by partitioning the workload in a modular
13 fashion. Each module operates as a Single Instruction
14 Stream Multiple Data stream (SIMD) computer system
15 composed of RISC processing elements, with each RISC
16 processor performing the functionality of a neuron
17 These modules are self contained units that can operate
18 as part of a multiple module configuration or work as
19 stand alone systems.

20

21 The hardware resources required to implement a module
22 have been minimised by applying modifications to the
23 original SOM algorithm. The key modification being the
24 replacement of the conventional Euclidean distance
25 metric by the simpler and easier to implement Manhattan
26 distance metric. The modifications made have resulted
27 in considerable savings of hardware resources because
28 the modular map design does not require conventional
29 multiplier units. The simplicity of this fully digital
30 design is suitable for implementation using a variety
31 of technologies such as VLSI or ASIC.

32

33 A balance has been achieved between the precision of
34 vector elements, the reference vector size and the
35 processing capabilities of individual neurons to gain
36 the best results for minimum resources. The potential

1 speedup of implementing all neurons in parallel has
2 also been maximised by storing reference vectors local
3 to their respective neurons (i.e. on chip as local
4 registers). To further support maximum data throughput
5 simple but effective parallel point to point
6 communications are utilised between modules. This
7 Modular Map design offers a fully digital parallel
8 implementation of the SOM that is scaleable and results
9 in a simple solution to a complex problem.

10

11 One of the objectives of implementing Artificial Neural
12 Networks (ANNs) in hardware is to reduce processing
13 time for these computationally intensive systems.
14 During normal operation of ANNs significant computation
15 is required to process each data input. Some
16 applications use large input vectors, sometimes
17 containing data from a number of sources and require
18 these large amounts of data processed frequently. It
19 may even be that an application requires reference
20 vectors updated during normal operation to provide an
21 adaptive solution, but the most computationally
22 intensive and time consuming phase of operation is
23 network training. Some hardware ANN implementations,
24 such as those for the multi-layer perceptron, do not
25 implement training as part of their operation, thereby
26 minimising the advantage of hardware implementation.
27 However, Modular Maps do implement the learning phase
28 of operation and, in so doing, maximise the potential
29 benefits of hardware implementation. Consequently,
30 consideration of the time required to train these
31 networks is appropriate.

32

33

34 Background

35

36 The modular approach towards implementation results in

1 greater parallelism than does the equivalent unitary
2 network implementation. It is this difference in
3 parallelism that has the greatest effect on reducing
4 training times for Modular Map systems. Consideration
5 was given to developing mathematical models of the
6 Modular Map and SOM algorithms for the purpose of
7 simulating training times of the two systems.

8
9 The Modular Map and SOM algorithms have the same basic
10 phases of operation, as depicted in the flowchart of
11 Fig. 14. When considering an implementation strategy
12 in terms of partitioning the workload of the algorithm
13 and employing various scales of parallelism, the
14 potential speedup of these approaches should be
15 considered in order to minimise network training time.
16 Of the five operational phases shown in Fig. 14, only
17 two are computationally intensive and therefore
18 significantly affected by varying system parallelism.
19 These two phases of operation involve the calculation
20 of distances between the current input and the
21 reference vectors of all neurons constituting the
22 network, and updating the reference vectors of all
23 neurons in the neighbourhood of the active neuron (i.e.
24 phases 2 and 5 in Fig. 14).

25
26 To facilitate investigation into the potential speedup
27 of Modular Map systems over the alternative unitary
28 networks and serial implementation, the model used was
29 based on the two computationally intensive phases of
30 operation mentioned above. This allows assessment of
31 the trends in training times while varying parameters
32 such as network size and vector size, and facilitating
33 an understanding of the relative training times for
34 different implementation strategies.

35
36

1 Training Times for Parallel Implementation

2
3 A simplified mathematical model of the Modular Map can
4 be constructed for the purpose of assessing training
5 times. The starting point for this model will be the
6 neuron, as it is the fundamental building block of the
7 Modular Map. When the neuron is presented with an
8 input vector $x = [\epsilon_1, \epsilon_2, \dots, \epsilon_n] \in \mathbb{R}^n$ it proceeds to
9 calculate the distance between its reference vector $m_1 =$
10 $[\mu_{11}, \mu_{12}, \dots, \mu_{1n}] \in \mathbb{R}^n$ and the current input vector
11 x . The distance calculation used by the Modular Map is
12 the Manhattan distance, i.e.

$$\text{Distance} = \sum_{j=1}^n |\xi_j - \mu_j|$$

13
14
15 where n = vector size.

16
17 The differences between vector elements are calculated
18 in sequence as while all neurons are implemented in
19 parallel, vector elements are not. To implement the
20 system utilising this level of parallelism is not
21 practical because it would require either 16 separate
22 processors per neuron, or a vector processor for each
23 neuron, so that the distances between all vector
24 elements could be calculated simultaneously. The
25 resources required to process all vector elements in
26 parallel would be substantially greater than the
27 requirements of the RISC neuron (Fig. 11) and would
28 greatly reduce the chances of implementing a Modular
29 Map on a single device. Consequently, when n
30 dimensional vectors are used, n separate calculations
31 are required.

32
33 If the time required by a neuron to determine the
34 distance for one dimension is taken to be t_d seconds and
35 there are n dimensions, then the total time taken to
36 calculate the distance between input and reference

1 vectors (d) will be nt_d seconds i.e. $d = nt_d$ (seconds).
2 The summation operation is carried out as the distance
3 between each element is determined and is therefore a
4 variable overhead dependent on the number of vector
5 elements, and does not affect the above equation for
6 distance calculation time. However, the value for t_d
7 will reflect the additional overhead of this summation
8 operation, as it will all variable overheads
9 proportional to vector size for this calculation. The
10 reason being that the distance calculation time (t_d) is
11 the fundamental timing unit used in this model. It has
12 no direct relationship to the time an addition or
13 subtraction operation will take for any particular
14 device; it is the time required to calculate the
15 distance for a single element of a reference vector
16 including all variable overheads associated with this
17 operation.

18
19 As all neurons are implemented in parallel the total
20 time required for all neurons to calculate Manhattan
21 distance will be equal to the time it takes for a
22 single neuron to calculate its Manhattan distance.
23 Once neurons have calculated their Manhattan distances
24 the active neuron has to be identified before any
25 further operations can be carried out. This process
26 involves all neurons simultaneously subtracting one
27 from their current distance value until one neuron
28 reaches a value of zero. As this process only
29 continues until the active neuron has been identified,
30 (the neuron with minimum distance) relatively few
31 subtraction operations are required.

32
33 Data generated during the training of Modular Maps for
34 the GRANIT application (discussed later) was used to
35 evaluate the overheads involved in finding the active
36 neuron. Fig. 15 is a graph of the activation values

1 (Manhattan distances) of the active neuron for the
2 first 100 training steps. The data was
3 generated for a 64 neuron Modular Map with 16 inputs
4 using a starting neighbourhood covering 80% of the
5 network. The first few iterations of the training
6 phase (less than 10) have a high value for their
7 Manhattan distances as can be seen from Fig. 15.
8 However, after the first 10 iterations there is little
9 variation for the distances between the reference
10 vector of the active neuron and the current input.
11 Thus, the average activation value after this initial
12 period is only 10, which would require only 10
13 subtraction operations to find the active neuron.
14 Consequently, there is a substantial overhead for the
15 first few iterations, but these will be similar for all
16 networks and can be regarded as a fixed overhead which
17 is not accounted for in the simple timing model used.
18 Throughout the rest of the training phase the overhead
19 of calculating the active neuron is insubstantial and
20 will be assumed to be negligible for the sake of
21 simplicity.

22
23 During the training phase of operation, reference
24 vectors are updated after the distances between the
25 current input and the reference vectors of all neurons
26 have been calculated. This process again involves the
27 calculation of differences between vector elements as
28 detailed above. Computationally this is inefficient
29 because these values have already been calculated
30 during the last operational phase. However, to have
31 used the previously calculated values would have
32 required an additional 16 bytes of local memory for
33 each neuron to store these values and to avoid the
34 additional resource overhead these values are
35 recalculated. After the distance between each element
36 has been calculated these intermediate results are then

1 multiplied by the gain factor. The multiplication
2 phase is carried out by an arithmetic shifter mechanism
3 which is placed within the data stream and therefore
4 does not require any significant additional overhead
5 (see Fig. 11). The addition of these values to the
6 current reference vector will have an impact on the
7 update time for a neuron approximately equivalent to
8 the original summation operation carried out to
9 determine the differences between input and reference
10 vectors. Consequently, the time taken for a neuron to
11 update its reference vector is approximately equal to
12 the time it takes to calculate the Manhattan distance,
13 i.e. d (seconds), because the processes involved are
14 the same (i.e. difference calculations and addition).
15 The number of neurons to have their reference vectors
16 updated in this way varies throughout the training
17 period, often starting with approximately 80% of the
18 network and reducing to only one by the end of
19 training. However, the time a Modular Map takes to
20 update a single neuron will be the same as it requires
21 to update all its neurons because the operations of
22 each neuron are carried out in parallel.

23
24 Kohonen states that the number of training steps
25 required to train a single network is proportional to
26 network size. So let the number of training steps (s)
27 be equal to the product of the proportionality constant
28 (k) and the network size (N) (i.e. Number of training
29 steps required (s) = kN). From this simplified
30 mathematical model it can be seen that the total
31 training time (T_{par}) will be the product of the number
32 of training steps required (s), the time required to
33 process each input vector (d), and the time required to
34 update each reference vector (d) i.e. Total training
35 time (T_{par}) = $2ds$ (seconds), but $d = nt_d$ and $s = kN$, so
36 substituting and rearranging gives:

1
2 $T_{\text{par}} = 2Nnkt_d$ - Equation 1.1
3

4 This simplified model is suitable for assessing trends
5 in training times and shows that the total training
6 time will be proportional to the product of the network
7 size and the vector size, but the main objective is to
8 assess relative training times. In order to assess
9 relative training times consider two separate
10 implementations with identical parameters, excepting
11 that different vector sizes, or network sizes, are used
12 between the two systems such that vector size n_2 is some
13 multiple (y) of vector size n_1 . If $T_1 = 2Nn_1 kt_d$ and T_2
14 $= 2Nn_2 kt_d$, then by rearranging the equation for T_1 , n_1
15 $= T_1/(2Nkt_d)$ but, $n_2 = yn_1 = y(T_1/(2Nkt_d))$. By
16 substituting this result into the above equation for T_2
17 it follows that:

18
19 $T_2 = 2N y(T_1/(2Nkt_d)) kt_d = yT_1$ - Equation 1.2
20

21 The consequence of this simple analysis is that a
22 module containing simple neurons with small reference
23 vectors will train faster than a network of more
24 complex neurons with larger reference vectors. This
25 analysis can also be applied to changes in network size
26 where it shows that training time will increase with
27 increasing network size. Consequently, to minimise
28 training times both networks and reference vectors
29 should be kept to a minimum as is done with the Modular
30 Map.

31
32 This model could be further expanded to consider
33 hierarchical configurations of Modular Maps. One of
34 the advantages of building a hierarchy of modules is
35 that large input vectors can be catered for without
36 significantly increasing the system training time.

1 This situation arises because the training time for a
2 hierarchy is not the sum of training times for all its
3 constituent layers, but the total training time for one
4 layer plus the propagation delays of all the others.
5 The propagation delay of a module (T_{prop}) is very small
6 compared to its training time and is approximately
7 equal to the time taken for all neurons to calculate
8 the distance between their input and reference vectors.
9 This delay is kept to a minimum because a module makes
10 its output available as soon as the active neuron has
11 been determined, and before reference vectors are
12 updated. A consequence of this type of configuration
13 is that a pipelining effect is created with each
14 successive layer in the hierarchy processing data
15 derived from the last input of the previous layer.

16

17

$$18 \quad T_{prop} = nt_d \quad - \quad \text{Equation 1.3}$$

19

20 All modules forming a single layer in the hierarchy are
21 operating in parallel and a consequence of this
22 parallelism is that the training time for each layer is
23 equal to the training time for a single module. When
24 several modules form such a layer in a hierarchy the
25 training time will be dictated by the slowest module at
26 that level which will be the module with the largest
27 input vector (assuming no modules are connected
28 laterally). As a single Modular Map has a maximum
29 input vector size of 16 elements and under most
30 circumstances at least one module on a layer will use
31 the maximum vector size available, then the vector size
32 for all modules in a hierarchy (n_h) can be assumed to be
33 16 for the purposes of this timing model. In addition,
34 each module outputs only a 2-dimensional result which
35 creates an 8:1 data compression ratio so the maximum
36 input vector size catered for by a hierarchical Modular

1 Map configuration will be 2×8^l (where l is the number
2 of layers in the hierarchy). Consequently, large input
3 vectors can be accommodated with very few layers in a
4 hierarchical configuration and the propagation delay
5 introduced by these layers will, in most cases, be
6 negligible. It then follows that the total training
7 time for a hierarchy (T_h) will be:

8

$$9 \quad T_h = 2Nn_hkt_d + (l-1)n_h t_d \approx 2Nn_hkt_d \quad - \quad \text{Equation 1.4}$$

10

11 By following a similar derivation to that used for
12 equation 1.2 it can be seen that:

13

$$14 \quad T_{par} \approx \gamma T_h \quad - \quad \text{Equation 1.5}$$

15

16 Where the scaling factor $\gamma = n/n_h$.

17

18 This modular approach meets an increased workload with
19 an increase in resources and parallelism which results
20 in reduced training times compared to the equivalent
21 unitary network and, this difference in training times
22 is proportional to the scaling factor between the
23 vector sizes (i.e. γ).

24

25

26 Training Times for Serial Implementation

27

28 The vast majority of ANN implementations have been in
29 the form of simulations on traditional serial computer
30 systems which effectively offer the worst of both
31 worlds because a parallel system is being implemented
32 on a serial computer. As an approach to assessing the
33 speedup afforded by parallel implementation the above
34 timing model can be modified. In addition, the
35 validity of this model can be assessed by comparing
36 predicted relative training times with actual training

1 times for a serial implementation of the Modular Map.

2

3 The main difference between parallel and serial
4 implementation of the Modular Map is that the
5 functionality of each neuron is processed in turn which
6 will result in a significant increase in the time
7 required to calculate the Manhattan distances for all
8 neurons in the network compared to a parallel
9 implementation. As the operations of neurons are
10 processed in turn there will also be a difference
11 between the time required to calculate Manhattan
12 distances and update reference vectors. The reason for
13 this disparity with serial implementation is that only
14 a subset of neurons in the network have their reference
15 vectors updated, which will clearly take less time than
16 updating all neurons constituting the network when each
17 reference vector is updated in turn.

18

19 The number of neurons to have their reference vectors
20 updated varies throughout the training period, starting
21 with 80% and reducing to only one by the end of
22 training. As this parameter varies with time it is
23 difficult to incorporate into the timing model, but as
24 the neighbourhood size is decreasing in a regular
25 manner the average neighbourhood size over the whole
26 training period covers approximately 40% of the
27 network. The time required to update each reference
28 vector is also approximately equal to the time required
29 to calculate the distance for each reference vector,
30 and consequently the time spent updating reference
31 vectors for a serial implementation will average 40% of
32 the time spent calculating distances. In order to
33 maintain simplicity of the model being used, the
34 workload of updating reference vectors will be evenly
35 distributed among all neurons in the network and,
36 consequently, the time required for a neuron to update

1 its reference vectors will be 40% of the time required
2 for it to calculate the Manhattan distance, i.e. update
3 time = 0.4d (seconds).

4

5 In this case equation 1.1 becomes:

6

7

8 $T_{\text{serial}} = 1.4 N^2 n k t_d$ (seconds) - Equation 1.6

9

10 This equation clearly shows that for serial
11 implementation the training time will increase in
12 proportion to the square of the network size.
13 Consequently, the training time for serial
14 implementation will be substantially greater than for
15 parallel implementation. Furthermore, comparison of
16 equation 1.1 and 1.6 shows that $T_{\text{serial}} = 0.7NT_{\text{par}}$, i.e.
17 the difference in training time for serial and parallel
18 implementation will be proportional to the network
19 size.

20

21 A series of simulations were carried out using a single
22 processor on a PowerXplorer system to assess the trends
23 and relationships between training times for serial
24 implementation of Modular Maps and provide some
25 evidence to support the model being used. The
26 simulations used a Modular Map simulator (MAPSIM) to
27 train various Modular Maps with a range of network and
28 vector sizes. As the model does not take account of
29 data input and output overheads these were not used in
30 the determination of training times, although the
31 training times recorded did include the time taken to
32 find the active neuron.

33

34 Some assumptions and simplifications have been
35 incorporated into this model, but have been
36 incorporated in such a way as to facilitate a good

1 approximation of timing behaviour. The simulations
2 that were run to help evaluate this model showed that
3 trends in training time did follow those prescribed by
4 equation 1.6 (see figure 16). Fig. 16 shows that the
5 range of training time required for a 99 element vector
6 increases substantially for increased network size,
7 whereas for a 16 element vector, the increase in
8 training time is not so substantial. When the actual
9 training time is known for one configuration, the
10 training times for other configurations can be
11 calculated using equation 1.2 and all predicted times
12 using this approach were within 10% of the actual
13 training time measured on the PowerXplorer.

14
15 The three main implementation strategies are serial
16 implementation, fine grain parallelism for a unitary
17 network and fine grain parallelism for a modular
18 network. Fig. 17 is a graph which has been constructed
19 to show the theoretical differences in training times
20 for these three strategies. The training times
21 presented for serial implementation have been derived
22 from actual training times measured on the PowerXplorer
23 and the other plots have been calculated relative to
24 these values using the model. Fig. 17 clearly
25 indicates that a modular approach to implementation
26 which utilises fine grain parallelism offers
27 considerably reduced training times compared to the
28 other strategies considered.

29
30 The model has been developed from the two
31 computationally intensive phases of operation that
32 involve the calculation of distances and updating of
33 reference vectors, as shown in Fig. 14. These are the
34 phases of operation that will be most affected by
35 increasing system parallelism and offer a good
36 approximation of timing behaviour.

1 Consideration could also be given to the overheads of
2 data input and output for these implementation
3 strategies although the impact of these overheads will
4 be minimal compared to the time required for the
5 computationally intensive phases of operation mentioned
6 above. The data output operation involves outputting
7 the XY coordinates of the active neuron for the Modular
8 Map. This approach could also be used for the other
9 implementation approaches considered here. The Modular
10 Map design allows the output to be made available as
11 soon as the coordinates of the active neuron have been
12 determined. Both output values are maintained at the
13 output of the device until they are read, but once the
14 output has been made available the other processes
15 continue, leaving the data transfer to be handled by an
16 autonomous handshake system. The same approach could
17 be adopted by a unitary network system, but serial
18 implementation would have to output the X and Y
19 coordinates separately and all other processing would
20 have to stop while these operations were being carried
21 out. This would result in the serial implementation
22 taking more time to perform data output than the other
23 two approaches, but the impact on overall training time
24 would be minimal.

25
26 The data input phase of operation requires more time
27 than does data output, but again the Modular Map design
28 aims to minimise the overheads involved. The Modular
29 Map will require a maximum of eight read cycles per
30 input vector because input vectors have a maximum of 16
31 elements and two of these elements are read on each
32 cycle. In addition, the inputs for Modular Maps are
33 buffered and most of these read cycles can be carried
34 out while previously read data is being processed by
35 the neural array. If the same approach were used for a
36 unitary network with larger input vectors, the

1 overheads would be similar because the neural array
2 would be processing previously read data while new data
3 was being input to the data buffer. Again it is the
4 serial implementation strategy that will suffer the
5 greatest overhead for this phase of operation because
6 each vector element has to be read in separately, and
7 while data is being input no other processing is able
8 to proceed. Consequently, serial implementation will
9 suffer a data input overhead proportional to the vector
10 size.

11

12 **Applications**

13

14 Modular Maps offer a versatile implementation of
15 Kohonen's Self-Organising Map (SOM) that is suitable
16 for use in a wide variety of problem domains. Two
17 possible application have been used as examples of the
18 applications for which Modular Maps are suited; human
19 face recognition and ground anchorage integrity
20 testing. The applications have little in common other
21 than their ill-defined nature but, Modular Maps offer
22 possible solutions in both domains. The SOM is also
23 applied to these problems to provide a benchmark for
24 the Modular Map approach.

25

26 Human face recognition is an ill-defined problem that
27 is difficult to tackle using conventional computing
28 techniques but has aspects that make it amenable to
29 solution by neural network systems. There are many
30 approaches to the face recognition problem that have
31 been attempted over the years utilising a range of
32 techniques including statistical and genetic algorithm
33 approaches. However, the aim here is to assess Modular
34 Maps as an alternative to the traditional SOM.
35 Consequently, comparisons are only made between the SOM
36 and Modular Map solutions.

1 As the SOM is the basis for the Modular Map design, the
2 classification and clustering of the two systems are
3 further compared in the application domain of ground
4 anchorage integrity testing (GRANIT). This is also an
5 application that is difficult to tackle using
6 conventional computing techniques, but its ill-defined
7 nature and high noise levels make it a suitable
8 application for a neural network solution. The
9 application is currently being developed at the
10 University of Aberdeen to provide an easy to use
11 mechanism to replace the current conventional test
12 procedures used within the civil engineering industry
13 which are time consuming, expensive and often
14 destructive.

15

16

17 Human Face Recognition

18

19 Human face recognition is generally regarded as a very
20 difficult task for computing systems to undertake.
21 There are databases containing face images available
22 via the Internet, e.g. the Olivetti web site but, like
23 many Internet resources, there is no standardisation
24 from one site to another. Consequently, it is
25 difficult to obtain a data set of face images in a
26 usable format containing sufficient variations and
27 instances of each face to enable training of ANN
28 systems. However, at the University of Aberdeen, Dr
29 Ian Craw of the Department of Mathematics has been
30 working in the field of face recognition for some time
31 and has built several face databases. Access to some
32 of this data was arranged, along with permission to use
33 it as part of the evaluation of Modular Map systems,
34 which avoided the problems of loading large data files
35 from the Internet.

36

1 The data base used for evaluation of Modular Maps was
2 derived from photographs of human faces taken by a
3 colour CCD camera connected to a framegrabber which
4 digitised colour at a resolution of 576 x 768 pixels.
5 A total database of 378 images made up from 14
6 photographs of 27 different subjects was created in
7 this way. The photographs were taken over a period of
8 weeks with varying intervals between shots using
9 differing lighting conditions and a variety of
10 orientations of the subject. Fig. 18 shows a typical
11 example of the types of images used in greyscale.
12 Excessive variation was avoided to prevent potential
13 matches based on condition rather than subject. None
14 of the photographs included faces with glasses or
15 beards but the clothing worn by subjects changed
16 throughout their series of photographs.

17
18 The background of the photographs was eliminated to
19 leave images of 128 x 128 pixels, but the hair which is
20 not invariant over time was left in the picture.
21 Thirty-four landmarks were then found manually for each
22 image to create a face model. The images are then
23 scaled ('morphed') to minimise the error between
24 landmark positions for individual images and a
25 reference face; the reference face being used here is
26 the average of the ensemble of faces. This process
27 normalises the images for inter-ocular distance and
28 ocular location (i.e. the faces are scaled and
29 translated to put the centre of both eyes in the same
30 X,Y location for all images). This normalisation
31 process removes the effects of different camera
32 locations and face orientations and offers an
33 alternative to positioning subjects carefully before
34 images are acquired. The average image is calculated
35 from the whole database and, in addition to being used
36 as detailed above, is subtracted from each image

1 resulting in a face subspace of $n-1$, where n was the
2 original dimensionality of the images.
3
4 Principal Component Analysis (PCA) may then be
5 performed separately on the shape-free face images and
6 the shape vectors consisting of the X,Y location of the
7 points on the original face image. The data used for
8 the evaluations used the shape-free face images. The
9 normalised images were considered as raster vectors and
10 subjected to PCA where the eigenvalues and unit
11 eigenvectors (eigenfaces of 99 elements) of the image
12 cross-correlation matrix were obtained. PCA has the
13 effect of reducing the dimensionality of the data by
14 "transforming to a new set of variables (principal
15 components) which are uncorrelated, and which are
16 ordered so that the first few components retain most of
17 the variation present in all of the original
18 variables". While PCA is a standard statistical
19 technique for reducing the dimensionality of data and
20 attempting to preserve as much of the original
21 information as possible it is difficult to give
22 meaningful labels to individual components.
23
24 Hancock and Burton have investigated principal
25 component representations of faces and suggest several
26 correlations with PCA components of shape vectors and
27 face features such as head size, nodding and shaking of
28 the head and variations in face shape. However, little
29 is suggested about the correlations between PCA
30 components derived from the shape-free vectors and face
31 features. It appears that individual PCA components
32 derived from shape free face images do not normally
33 correlate directly to individual face features, but the
34 first two components of the eigenface are believed to
35 be associated with the size of the face and lighting
36 conditions. It is because of the application that

1 these eigenvectors are often referred to as eigenfaces.

2

3 It was these eigenfaces that were made available for
4 the Modular Map investigation. In ANN terms this
5 database contained a very limited dataset and, normally
6 many more than 14 instances of a class would be used to
7 train a network. However, this still offered an
8 improvement over other sources such as the Olivetti
9 data base which only had 10 instances of each face. To
10 facilitate both training and testing of ANN systems
11 nine eigenfaces for each subject were used to train a
12 network and the other five were used to test its
13 classification. The test set was selected across the
14 range of orientation and lighting conditions so that
15 the training set would also cover the whole range of
16 conditions.

17

18 The eigenface data consisted of double precision
19 floating point values between minus one and plus one
20 but Modular Maps only accept eight bit inputs.
21 Consequently, the face data needed to be converted to
22 suitable eight bit values before it could be used with
23 Modular Map systems. This was achieved using some
24 utility programs developed for use with Modular Map
25 systems. This software was able to offset data values
26 so that all values were positive, scale the data to
27 cover the range 0 to 255 and convert it to integer
28 (8 bit) values. The effects of this data manipulation
29 do not change the relationships between vector elements
30 as the same scaling and offset are applied to each
31 element but, rounding does occur during the conversion
32 process. It is also perhaps noteworthy that all data
33 used in the training and testing of a network should
34 use the same scaling factor and offset values to
35 maintain its integrity.

36

1 To facilitate the training and testing of neural
2 networks the eigenface data was split into nine
3 training vectors and five test vectors for each face.
4 To ensure that the networks were trained on the whole
5 range of possible orientations and lighting conditions
6 the first two and last two vectors in a class were
7 always used for training. The rest of the data was
8 selected as training vectors and test vectors
9 alternately such that on one simulation eigenfaces 1,
10 2, 4, 6, 8, 10, 12, 13 and 14 were used to train the
11 network while eigenfaces 3, 5, 7, 9 and 11 were used to
12 test the network. The next simulation would then use
13 eigenfaces 1, 2, 3, 5, 7, 9, 11, 13 and 14 to train the
14 network and eigenfaces 4, 6, 8, 10 and 12 to test the
15 network etc.

16

17

18 Using Kohonen's Self Organising Map to Classify Face 19 Data

20

21 Simulations using Kohonen's Self Organising Map (SOM)
22 were carried out to provide a benchmark for the Modular
23 Map evaluation. The first of these simulations used
24 the original double precision floating point data and a
25 64 neuron SOM, but the majority of vectors caused the
26 activation of the same neuron. Investigation found
27 that the problem was that the original data set
28 actually covered a smaller range than had been expected
29 and required excessive precision with regard to the ANN
30 processes. Rather than the data covering the whole
31 range between minus one and plus one, most vector
32 elements had a maximum variance of less than 0.1 over
33 the entire data set and the maximum variance found for
34 any element was less than 0.7. Consequently, it was
35 possible to have vectors originating from different
36 faces with a Euclidean distance much less than one.

1 The SOM implementation used double precision values
2 but, rounding errors within the mechanism resulted in
3 problems with the original data set.

4
5 Due to the problems encountered with the original
6 eigenfaces, the data was scaled to cover the range
7 between 0 and 255 but, using floating point values
8 rather than the 8 bit data required for Modular Maps.
9 When the 135 test vectors were presented to the network
10 this approach proved to offer much better results but,
11 high classification error rates of 40% were still
12 encountered (i.e. of the 135 test vectors presented to
13 the network after training, only 81 (60%) were
14 correctly identified). The reason for this poor
15 performance was that each class of data caused the
16 activation of several neurons and there were simply not
17 enough neurons in the network for all activation
18 regions to be distinct (i.e. a larger network was
19 required). Fig. 19a is an example activation region
20 for a modular map and Fig. 19b is an example activation
21 map for a SOM. When the same data was used with a SOM
22 network of 256 neurons the error rate dropped to 6%.
23 When simulations were run using a quantised version of
24 the data set (i.e. using integer values) the results
25 were found to be identical thereby suggesting that the
26 rounding errors within the data introduced by the
27 quantisation process were not significant (see the
28 error rate table (table 1 below).
29

ANN type	Configuration Details	% Error
SOM	64 Neurons Floating point data (99 element vectors)	40 \pm 12
SOM	64 Neurons Integer data (99 element vectors)	40 \pm 12
SOM	256 Neurons Floating point data (99 element vectors)	6 \pm 1
SOM	256 Neurons Integer data (99 element vectors)	6 \pm 1
SOM	1024 Neurons Floating point data (99 element vectors)	6 \pm 1
SOM	256 Neurons Floating point data Using overlap data (127 element vectors)	7 \pm 1
Modular Map	Nine Module Hierarchy 7 with 13 inputs 1 with 8 inputs Output = 64 Neurons (configuration 1)	19 \pm 3
Modular Map	Seven Module Hierarchy 6 with 16 inputs Output = 64 Neurons (configuration 2)	18 \pm 3
Modular Map	Nine Module Hierarchy Using overlap data 7 with 16 inputs, 1 with 15 inputs Output = 64 Neurons (configuration 3)	11 \pm 2
Modular Map	Nine Module Hierarchy Using overlap data 7 with 16 inputs, 1 with 15 inputs Output = 256 Neurons (configuration 4)	4 \pm 1

Table 1 Summary Classification Error Rate Table. .
 Figures quoted are mean classification errors
 with standard deviation. All figures are
 quoted to the nearest integer value.

1
2 Using Modular Maps to Classify Face Data

3
4 Modular Maps can be combined in different ways and use
5 different data partitioning strategies. Four separate
6 Modular Map configurations are used to outline the
7 effects of using different approaches. The first
8 approach to Modular Map solution of the eigenface
9 classification problem presented is intended more as a
10 'how not to do' approach. This combination of modules,
11 configuration 1, utilises nine Modular Map networks
12 each with 64 neurons (see Fig. 20). The topology of
13 the system is hierarchical with eight modules at the
14 base of the hierarchy (the input layer I) and one at
15 the output level (output layer O). The data was
16 partitioned so that seven modules each had 13 inputs
17 and one module had 8 inputs. This data partitioning
18 strategy may result in poor classification because a
19 module will give better results when the whole of the
20 reference vector is utilised (i.e. when all 16 inputs
21 are used).

22
23 The results from simulations using configuration 1
24 (Fig. 20) showed poor classification of the face data
25 with an average classification error of 19% from the
26 output module. It can also be seen from table 2 below
27 that the error rate for module 7, which only has eight
28 inputs as opposed to the 13 used by all other networks
29 at that level, are much higher than all other networks.

30
31 A factor contributing to this is that module 7 has much
32 fewer inputs, which will naturally lead to poorer
33 performance but, it should also be noted that there is
34 a general trend of classification errors from modules
35 at the base of the hierarchy which correlates to the
36 importance of the elements of the eigenvectors (i.e.

the first few PCA elements have most of the variation). However, the small number of vector elements used is the most prominent factor contributing to poor performance and this is highlighted by the results of configuration 2 (Fig. 21) which show considerably better classification results for most modules at the base of the hierarchy when all 16 inputs are used.

Module	No of Inputs	% Error
0	13	20
1	13	22
2	13	21
3	13	21
4	13	28
5	13	29
6	13	29
7	8	39
8	16	19

Table 2 : Error Rate Table for Configuration 1 (Fig. 20)

The second Modular Map configuration (configuration 2 shown in Fig. 21) used only seven modules in total; six on the input layer I of the hierarchy and one at the output layer O. The data was partitioned so that all modules at the base of the hierarchy had sixteen inputs, which gives a total of 96 input vector elements as opposed to the 99 in the original eigenfaces; the final three elements of the eigenfaces being the least significant ones and therefore omitted.

1 The results from this series of simulations showed an
2 improved classification but, only an increase of 1% on
3 the previous error rates for the output module were
4 achieved (table 3 below). The overall performance
5 increase is due in part to the fact that the output
6 module is now only using 12 out of the 16 possible
7 inputs. However, most modules had reduced error rates
8 compared to the previous series of simulations and all
9 modules had better classification rates than had been
10 experienced for module 7 in configuration 1 (Fig. 20).
11 An additional two modules could be added to the base of
12 the hierarchy so that the output module would be using
13 all of its inputs. One possible approach would be to
14 simply present the first 16 elements of the eigenfaces
15 to two modules. This type of approach is normally
16 referred to as an ensemble and has been found to
17 improve classification. There are no known
18 dependencies between vector elements of the eigenfaces
19 and there is no direct correlation between individual
20 elements and particular face features so the data
21 overlap approach was used to spread the data being used
22 for two inputs across the whole vector rather than
23 relying solely on any one block of 16 elements.
24

Module	No of Inputs	% Error
0	16	21
1	16	20
2	16	21
3	16	22
4	16	25
5	16	25
6	16	28
7	14	18

Table 3 : Error Rate Table for Configuration 2 (Fig. 21)

Utilising all inputs for modules at the base of the hierarchy improves classification. To maximise on this and the number of inputs to the next layer of the hierarchy, some of the input vector elements can be fed to more than one module. This 'data overlap' technique is where the data is split into groups of 16 element inputs, but the last few elements of one input vector are also used as inputs for the next module. This was accomplished by feeding vector elements 0 to 15 to module 0 and, elements 12 to 27 to module 1 etc. so that there was effectively an overlap of four vector elements between modules. In this way modules 0 to 6 all had 16 inputs but, module 7 only had 15 because when using the original 99 element vectors this was the closest to maximum input usage that could be achieved without using different strategies for different modules. This approach was chosen because it enables most modules at the base of the hierarchy to have 16 inputs and therefore helps to maximise the limited

1 amount of training data.

2

3 As with the first configuration, a total of nine
4 modules all with 64 neurons were used and were
5 connected together in a hierarchical manner as shown in
6 Fig. 22. The simulations carried out using this 'data
7 overlap' approach showed a significant improvement over
8 configurations 1 and 2 (Figs 20 and 21) because the
9 classification error from the output module had been
10 reduced to 11%. However, the classification errors for
11 modules at the base of the hierarchy did not show any
12 significant statistical difference to those found with
13 configuration 2 (Fig. 21) (compare table 3 and table 4
14 below). This suggests that the improvement in
15 classification is not due to the particular
16 partitioning strategy used, but to the fact that more
17 inputs to the hierarchy were used.

18

19

20

21

22

23

24

25

26

27

28

29 Table 4 : Error Rate Table for Configuration 3 (Fig.

30 22)

31

Module	No of Inputs	% Error
0	16	21
1	16	20
2	16	19
3	16	21
4	16	24
5	16	24
6	16	26
7	15	28
8	16	11

1 From the simulations performed using the SOM it was
2 noted that the activation regions for the face data
3 were such that a 256 neuron SOM was required to
4 classify the data with reasonable accuracy. The
5 simulations carried out using Modular Maps for this
6 data found that fewer neurons were active on the output
7 module of a Modular Map hierarchy than for the SOM.
8 This occurs because of the data compression being
9 performed by successive layers in the hierarchy and
10 results in a situation where fewer neurons are required
11 in the output network of a hierarchy of Modular Maps
12 than are required by a single SOM for the same problem.
13 However, when only a two layer hierarchy is being used
14 the compression is not sufficient for a 256 neuron
15 module to be replaced by a 64 neuron module. In
16 addition, Modular Maps can be combined both laterally
17 and hierarchically to provide the architecture suitable
18 for numerous applications.

19
20 Configuration 4 (Fig. 23) has 256 neurons at the output
21 layer 0 of a Modular Map hierarchy but all other
22 modules in the system were still maintained at 64
23 neurons. To create an array of 256 neurons, four
24 Modular Maps are connected together in a lateral
25 configuration and because modules connected in this way
26 act as though they were a single Modular Map they can
27 then be further combined to create hierarchies
28 containing different sized networks.

29
30 For these simulations the input data and the eight base
31 modules were identical to those detailed for
32 configuration 3 (Fig. 22); the only change was to the
33 size of the output module. The results of these
34 simulations showed that the classification error at the
35 output of the hierarchy had been reduced to 4% (the
36 results from layer one being identical to those for

1 configuration 3) which offered an improvement over all
2 previous simulations, including the ones using the
3 standard Kohonen network.

4

5

6 ANN Classification of Faces

7

8 The hardware required to provide the Modular Map
9 solution for this face recognition problem would
10 comprise 12 modules which could be implemented on
11 twelve VLSI devices. The SOM solution, however, would
12 require a network of 256 neurons, each capable of using
13 reference vectors of 99 elements. The digital hardware
14 requirements for a parallel implementation of such a
15 SOM would not fit onto a single VLSI device and would
16 require wafer scale integration for a monolithic
17 implementation. Even when attempting to implement this
18 SOM on several separate devices there are no known
19 systems with a comparable level of parallelism to the
20 Modular Map solution outside the realms of
21 neuro-computers and super-computers. There are, of
22 course, many other ways of implementing a SOM of this
23 size, e.g. transputer systolic array, but at present
24 the difficulties of implementing this comparatively
25 small SOM network on a single device in digital
26 hardware have been sufficient to prevent its
27 occurrence.

28

29 The results of these simulations show that Modular Maps
30 can be combined in a hierarchical and/or lateral
31 configuration to good effect. It was also shown that
32 to maximise the classification potential of Modular Map
33 hierarchies all inputs to modules should be used.
34 There are a variety of possible approaches
35 to maximising inputs and in this case a 'data overlap'
36 approach was used to maximise the limited training data

1 available and thereby improve classification results.
2
3 It was also found that the Modular Map approach to
4 classification of this face data offers slightly better
5 classification than the traditional SOM (see the
6 summary error rates table 1). In addition, the
7 clustering on the surface of output modules was
8 improved over that found on the SOM as can be seen from
9 the activation maps presented in appendix A. When
10 using a Modular Map hierarchy in configuration 4 (Fig.
11 23) the output module averaged 147 inactive neurons
12 compared to 106 for the 256 neuron SOM, the reason
13 being that the number of neurons active for individual
14 classes is reduced (i.e. tighter clustering is found on
15 the surface of the map). The clustering produced by
16 the Modular Map systems is similar to that of the SOM,
17 but was generally better defined. This can be seen
18 when comparing the neural activations created by the
19 same single class for the two systems, an example of
20 which is presented in Figs 19a and 19b. This example
21 corresponds to the activations for data class 3 in
22 appendix A. These differences are due to the different
23 architectures of the two systems. The SOM will only
24 have a single reference vector (containing 99 elements
25 in this case) while a Modular Map hierarchy results in
26 reference vectors for the output neurons being
27 constructed from a number of reference vectors from
28 lower levels in the hierarchy (effectively providing
29 127 elements here). Because the reference vectors of
30 the output layer of a Modular Map hierarchy are
31 constructed from several lower level reference vectors
32 it is possible to represent complex regions of the
33 feature space with few neurons at the output.
34
35 The Modular Map solution to the face recognition
36 problem requires more neurons than does the SOM

1 solution, but the RISC neurons used by Modular Maps are
2 much simpler which will result in a much reduced
3 resource requirement when implemented in hardware as
4 intended. It is the architecture of the Modular Map
5 approach that has resulted in better classification
6 rather than the number of neurons. This is emphasised
7 by the failure of the SOM to improve over the
8 previously stated classification results when network
9 size is increased beyond 256 neurons. When a SOM
10 containing 1024 neurons was trained on the same data
11 detailed above for the face recognition problem, the
12 classification of this data still resulted in a 6%
13 error for the test data. Simulations were also carried
14 out to check that the 'data overlap' approached used
15 for the Modular Map hierarchy shown in configuration 4
16 (Fig. 23) was not giving the Modular Map solution an
17 unfair advantage. These simulations used the same data
18 as had been used for the Modular Map configuration
19 except that the separate input vectors for modules were
20 joined together to form 127 element vectors (i.e. $7 \times$
21 $16 + 1 \times 15$ vector elements). When a 256 neuron SOM
22 was trained using these 127 element vectors equivalent
23 to the 'data overlap' used for configuration 4 (Fig.
24 23), the classification results did not improve, but
25 resulted in an additional 1% error compared to
26 simulations using the 99 element vectors, i.e.
27 classification error was 7% (see the summary error
28 table 1).

29
30 In addition, the eigenface data used in the above face
31 recognition were derived using Principal Component
32 Analysis (PCA) which reduced the dimensionality of the
33 original pictures by transforming the original
34 variables into a new set of variables (the principal
35 components) in a way that retains most of the variation
36 present in the original data. The principal components

1 are ordered so that the first few dimensions retain
2 most of the variation present in all of the original
3 variables. The data presented to the modular map array
4 maintained this order such that module 0 in a hierarchy
5 had the first few dimensions and the highest indexed
6 module on the lowest level had the last few dimensions
7 etc. While the error rates of modules on the lowest
8 layer in a hierarchy do not show a monotonic increase
9 in error rate with increasing index, the general trend
10 shows that error rates increase as the PCA components
11 show decreasing variance.

12
13 When combining Modular Maps in hierarchical
14 configurations, the error rates at the output network
15 were less than those found for any modules at lower
16 levels in the hierarchy (see tables 2, 3 and 4). Both
17 classification and clustering improve moving up through
18 subsequent layers in a Modular Map hierarchy as though
19 higher layers in the hierarchy were performing some
20 higher level functionality.

21 22 23 Ground Anchorage Integrity Testing

24
25 The Ground Anchorage Integrity Testing System (GRANIT)
26 is being developed as a joint project between the
27 Universities of Aberdeen and Bradford in collaboration
28 with AMEC Civil Engineering Ltd. This work is built on
29 the research of Prof. A.A. Rodger and Prof. G.S.
30 Littlejohn into the effects of close proximity blasting
31 to rock bolt behaviour.

32
33 As part of this development process, field trials were
34 carried out at the Adlington site of AMEC Civil
35 Engineering Ltd. Two test ground anchorages were
36 installed by AMEC Civil Engineering Ltd for the purpose

1 of these trials. The analysis pertains to a single
2 strand anchor which has a diameter of 15.2mm, a total
3 length of 10m and a bond length of 2m. The drilling
4 records for this anchorage show that the soil
5 composition was weathered sandstone between 5m and 5.8m
6 with strong sandstone between 5.8m and 9.95m. Using a
7 pneumatic impact device to apply an impulse vibration
8 was initiated within the anchorage system. An
9 accelerometer affixed to the anchorage strand was then
10 used to detect vibrations within the system.

11
12 The accelerometer output was fed, via a charge
13 amplifier, to a notebook PC where the signals were
14 sampled at 40 kSamples/Sec by a National Instruments
15 DAQ 700 data acquisition card controlled by the GRANIT
16 software developed at the University of Aberdeen. This
17 software was developed using National Instruments
18 LabWindows/CVI and the C programming language. The
19 intricacies of data sampling and signal pre-processing
20 are handled by the DAQ 700 software and Labwindows.
21 However, laboratory tests using known signals were
22 carried out to check that signals were being captured
23 and processed as expected and no problems were
24 identified.

25
26 Data was gathered for five pre-stress levels of the
27 ground anchorage system; four of these levels were
28 known to be 10kN, 20kN, 30kN and 40kN values, while the
29 fifth level was initially unknown and used as a blind
30 test to evaluate the potential predictive capacity of
31 the GRANIT system. After results of the data analysis
32 were presented to AMEC Civil Engineering the pre-stress
33 value of the anchorage when the blind data were
34 generated was revealed to be approximately 18 kN.
35 Fifty (50) waveforms containing 512 samples were taken
36 at each level. Throughout this evaluation process the

1 blind test data were used only as a check; they were
2 not taken into account when determining statistics of
3 the main data set etc.

4
5 The time domain signals generated by the ground
6 anchorage approximate a damped impulse response (see
7 Figs 24a to 24e) and the envelope of these signals
8 often provides an indication of the pre-stress level of
9 the anchorage. Figs 24a to 24e show the average time
10 domain signals for the 10kN, 20kN, 30kN, 40kN and blind
11 tests respectively. However, the power spectra of
12 these signals provides a better insight into varying
13 pre-stress levels, and offers a significant compression
14 of the data by transforming the original 512
15 dimensional time domain signals into their frequency
16 components which, in this instance, resulted in 64
17 components. A 5th order Butterworth low pass filter
18 with a threshold of 5kHz was used to remove unwanted
19 high frequency components. The power spectrum of these
20 signals provides the average frequency components over
21 the entire signal and shows that power spectra vary for
22 varying pre-stress levels in the ground anchorage.
23 Manual comparison of the power spectra can be
24 difficult, but can be used to provide an approximation
25 of pre-stress levels (see Figs 25a to 25e). Figs 25a
26 to 25e show the average power spectrum for the 10kN,
27 20kN, 30kN, 40kN and blind tests respectively.
28 Analysis utilising wavelet transforms could be used to
29 provide a more detailed time-frequency analysis but the
30 power spectra data offers considerable compression over
31 the original input data and provided sufficient
32 information for this analysis.

33

34

35 **Classification of Ground Anchorage Pre-Stress Levels**
36 **Using the Self-Organising Map**

1 A 64 neuron SOM was trained using the 64 dimensional
2 power spectra derived from response signals of the
3 ground anchorage generated at known pre-stress levels.
4 The activation map was then derived after training was
5 complete by feeding test data to the network and noting
6 which neuron was active for which class of data.
7 However, this labelling process can be time consuming
8 when carried out manually so a small utility program
9 was developed which takes the output from the network
10 and calculates the activation map automatically by
11 correlating the original class of inputs with the
12 resultant neuron activation. Once the activations on
13 the surface of the map had been determined, the blind
14 data set was fed to the SOM and the resultant
15 activations were recorded and can be seen in Fig. 26.
16 All 50 samples gathered during the blind field test
17 caused the activation of neurons associated with the
18 20kN data class.

19
20 The grouping of activations (clustering) on the surface
21 of the SOM does not show a gradual transition from low
22 to high pre-stress levels moving across the surface of
23 the map (see Fig. 26). However, in most cases, there
24 is a clear distinction between activations for
25 different pre-stress levels, with very few neurons
26 being active for two or more pre-stress values. There
27 are regions of activation on the surface of the map
28 that can be assigned to known pre-stress values of the
29 anchorage but no individual pre-stress level has a
30 single distinctive cluster of activations. There are
31 several reasons for this, one of which is that data
32 sets were not as consistent as would have been desired,
33 especially the 30 and 40 kN cases. One factor that is
34 responsible for these inconsistencies is that the
35 impact applied to the anchorage varied slightly
36 throughout the testing period. However, the activation

1 map created from this data (Fig. 26) shows that the
2 active neurons for the blind data set correspond to
3 neurons which were active for the 20kN data set.
4 Consequently, it can be stated that the closest
5 matching pre-stress value to the blind data set is 20
6 kN.

7

8

9 **Classification of Ground Anchorage Pre-Stress Levels**
10 **Using Modular Maps**

11

12 A simple Modular Map configuration was used with the
13 ground anchorage data detailed above to show that
14 Modular Map hierarchies give improvements in
15 classification and clustering moving up the hierarchy.
16 A total of five modules were employed in a hierarchical
17 configuration as shown in Fig. 27. As the data
18 consisted of 64 dimensional vectors, each of the
19 original vectors were partitioned into four separate
20 vectors of 16 elements. The data were also scaled and
21 quantised to fulfil the input requirements of Modular
22 Maps but, in order to keep the configuration as simple
23 as possible no attempts were made to create an optimal
24 solution to the ground anchorage integrity testing
25 problem and no data overlapping was used.

26

27 When the Modular Map system was trained on the same
28 power spectra data of ground anchorage response signals
29 as the SOM (see Figs 25a to 25e), the resultant
30 activation maps for modules at the base of the
31 hierarchy show poor classification and clustering of
32 the blind data set (see Figs 28 to 31). The unknown
33 pre-stress value could not be determined correctly from
34 any individual one of these activation maps and, it is
35 also unlikely that it could be identified by manual
36 inspection of any combination of lower level maps.

1 However, all 50 samples of the blind test data set
2 caused the activation of neurons associated with the
3 20kN data on the output module of the hierarchy, as had
4 occurred with the SOM (see Fig. 32) showing that
5 classification does indeed improve moving up through a
6 modular map hierarchy.

7
8 In addition, identification of each data class required
9 fewer neurons in the output module of the hierarchy
10 than had been required for the SOM. Instead of the
11 three neurons that were active for the 20kN data on the
12 SOM (see Fig. 26). This class of data only resulted in
13 two active neurons for the Modular Map. As the Modular
14 Map system had fewer active neurons for each data class
15 than did the SOM, there were 24 inactive neurons and,
16 consequently, a 40 neuron module could have been used
17 in place of the 64 neuron module. This effect was also
18 found to increase as the depth of hierarchy increases
19 such that the disparity between the number of neurons
20 required by the SOM and the output module of a
21 hierarchy increases with increasing depth of hierarchy.
22 There are still similarities between the activations
23 formed by the SOM and Modular Map for this data, with
24 each class accounting for approximately the same
25 percentage of activations for both systems, suggesting
26 that the essential features of the data have been
27 maintained. Overall the Modular Map also has fewer
28 clusters (regions of activation) per class, than does
29 the SOM, thereby reducing the disjoint nature of
30 activation sets. For example, on the SOM the 30kN case
31 has three separate clusters and the 40 kN case has four
32 separate clusters but, the Modular Map has two and
33 three clusters for this data respectively.

34

35

36 The Modular Map approach to face recognition results in

1 a hierarchical modular architecture which utilises a
2 'data overlap' approach to data partitioning. When
3 compared to the SOM solution for the face recognition
4 problem, Modular Maps offer better classification
5 results. This improvement in classification is
6 achieved because a modular architecture is used.
7 Modular Maps provide the basic building block for
8 modular architectures and can be combined both
9 laterally and hierarchically to good effect as has been
10 shown.

11

12 When hierarchical configurations of Modular Maps are
13 created the classification at the output layer offers
14 an improvement over that of the SOM because the
15 clusters of activations are more compact and better
16 defined for modular hierarchies. This clustering and
17 classification improves moving up through successive
18 layers in a modular hierarchy such that higher layers,
19 i.e. layers closer to the output, effectively perform
20 higher, or more complex, functionality.

21

22 Application solutions using a modular approach based on
23 the Modular Map will result in more neurons being used
24 than would be required for the standard SOM. However,
25 the RISC neurons used by Modular Maps require
26 considerably less resources than the more complex
27 neurons used by the SOM. The Modular Map approach is
28 also scaleable such that arbitrary sized networks can
29 be created whereas many factors impose limitations on
30 the size of monolithic neural networks. In addition,
31 as the number of neurons in a modular hierarchy
32 increases, so does the parallelism of the system such
33 that an increase in workload is met by an increase in
34 resources to do the work. Consequently, network
35 training time will be kept to a minimum and this will
36 be less than would be required by the equivalent SOM

1 solution, with the savings in training time for the
2 Modular Map increasing with increasing workload.
3
4 Modifications and improvements may be made to the
5 foregoing without departing from the scope of the
6 present invention. Although the above description
7 describes the preferred forms of the invention as
8 implemented in special hardware, the invention is not
9 limited to such forms. The modular map and
10 hierarchical structure can equally be implemented in
11 software, as by a software emulation of the circuits
12 described above.
13

Appendix A

Sample Activation Maps

The activation maps presented in this appendix were derived from the application of human face recognition detailed in chapter 7. This application had 27 separate classes, i.e. there were pictures of 27 humans. Each square on the activation map represents a single neuron. When a neuron has activations for a particular class, the class number is denoted. Where no class number is denoted the neuron is not associated with any class, i.e. it has no activations.

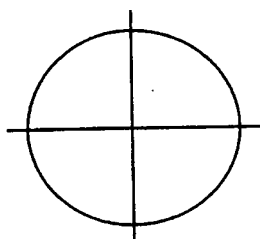
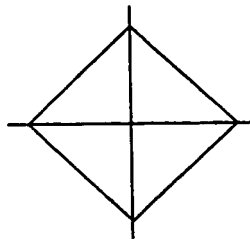
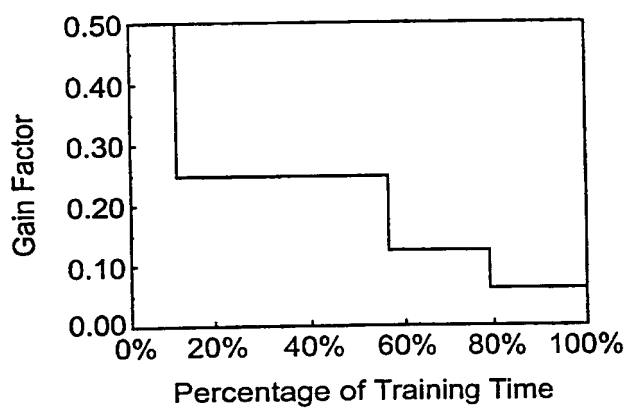
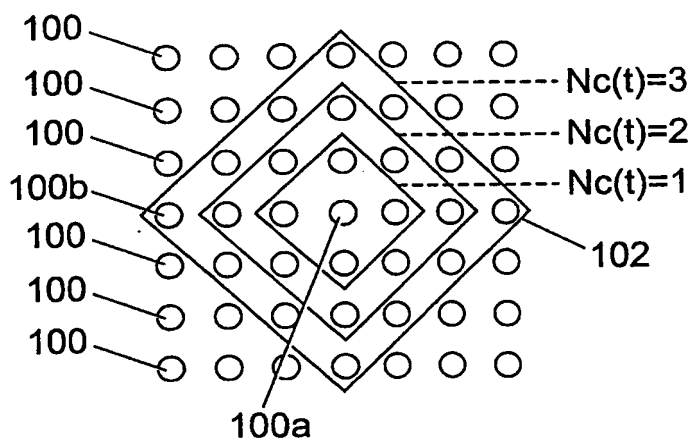
4			15	15	15	11	16	16						13	13
4	4		15			11	11	16						13	13
4	6	6					23	23	10	10				13	
4		6	6					23		10	10			12	21
	6	6		9	9	9		23	10		12			12	21
5		19	19			9			2	2		12			21
5	5	5		19	25		25		2					21	21
20	20					25		12	12	2	2	7	7	7	
1	18		18			15			12				7	26	
1	1	18		18	18	15		14	14		14			26	
27	27	1	1	18		15		14	14	14	14			26	19
27	27	18	18	16	16	11							26	26	19
20	22	22	22	16			11		17					26	19
20		22	3		11	11		17					8		24
		9		3				17	17	8			8	7	24
9	9	9		3	3	3	17	17			8	8	7		24

Figure A.1: Example activation map for a 256 neuron SOM trained on eigenface data

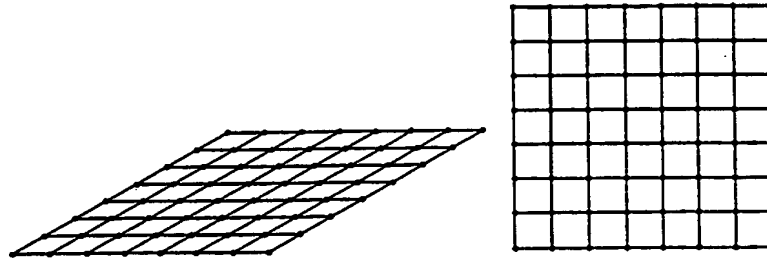
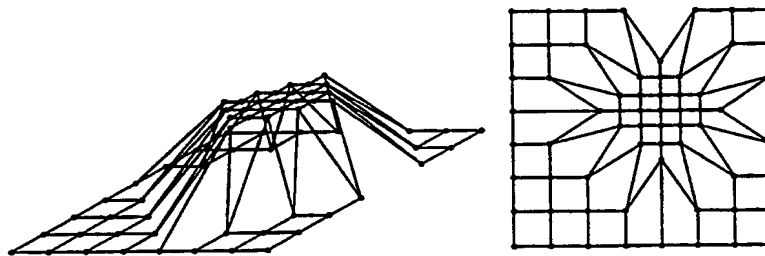
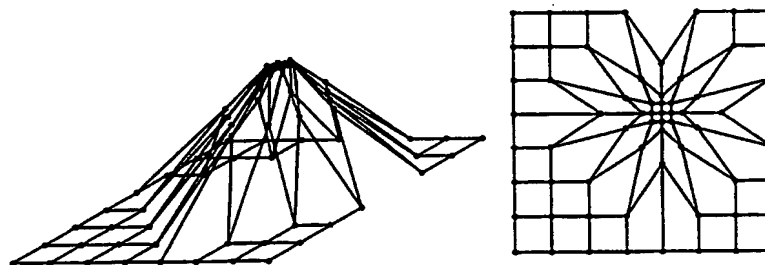
24			3				21		21			4	4		
			3	3				21				4	4		
						7		7				4			
26				11			7		7						6
	26		11		11		17	17				8			6
19	26	26		12		13	13		17			8		6	
19			14	12	13	13		17				8		6	6
		14	14			13		17		12					
		14	14						12		12	15			
	22	22	22					5	5	5	15	15			
								11		5		15			
1			25	27	27	27	27		11			2	20		
1			25	9			19	23				2	2	2	20 20
	1	1	18		9		19	23	23				2	20	
		18	18	18	9	9		16		23	10	10			
	18		18					16	16		10	10	10		

Figure A.2: Example activation map for a Modular Map Hierarchy (Configuration 4) trained on eigenface data

1 / 16

*Fig. 1a**Fig. 1b**Fig. 2**Fig. 3*

2 / 16

*Fig. 4a**Fig. 4b**Fig. 4c*

3 / 16

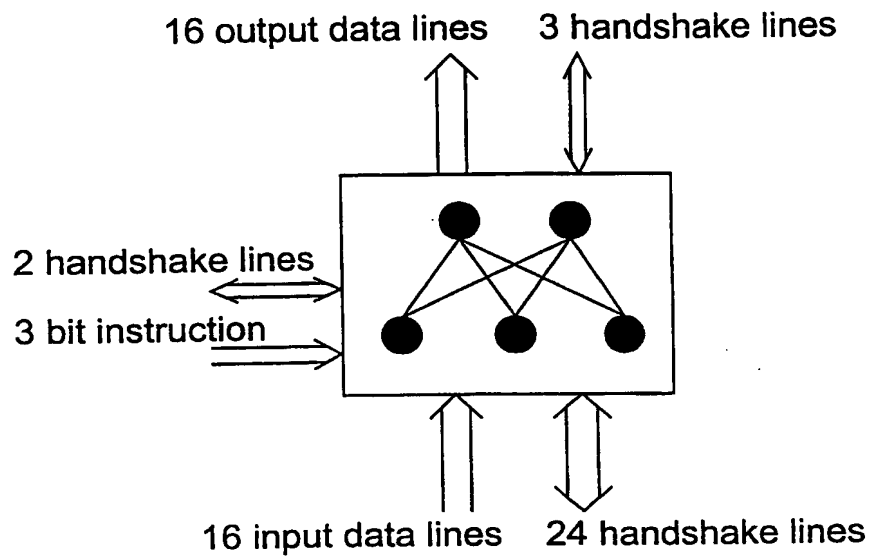


Fig. 5

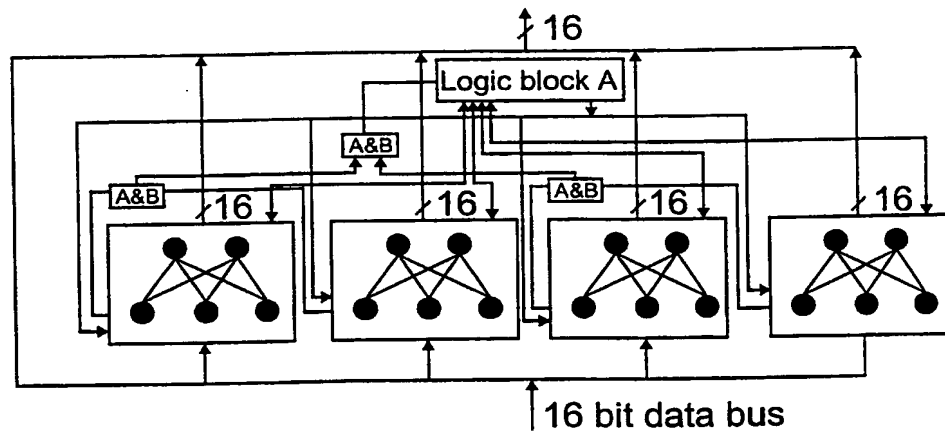


Fig. 6

4 / 16

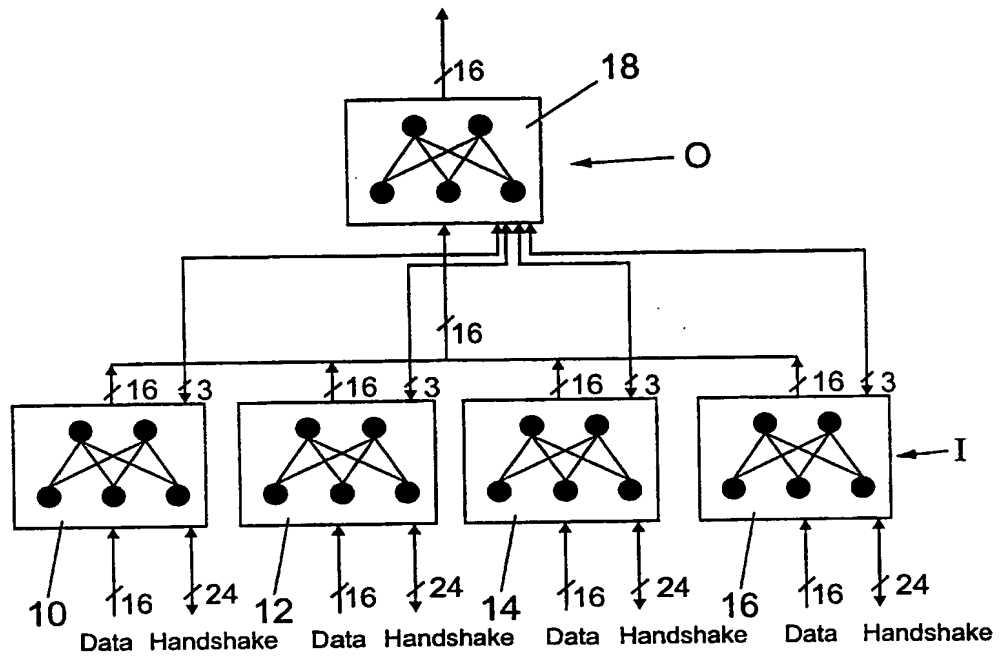


Fig. 7

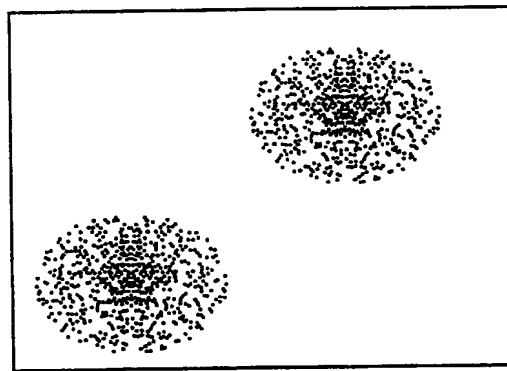
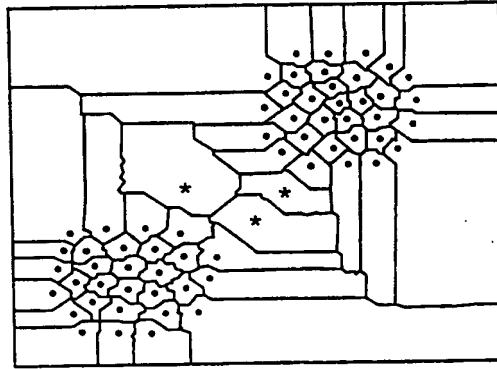
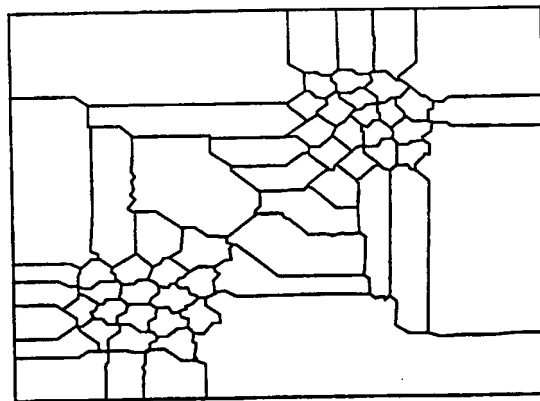
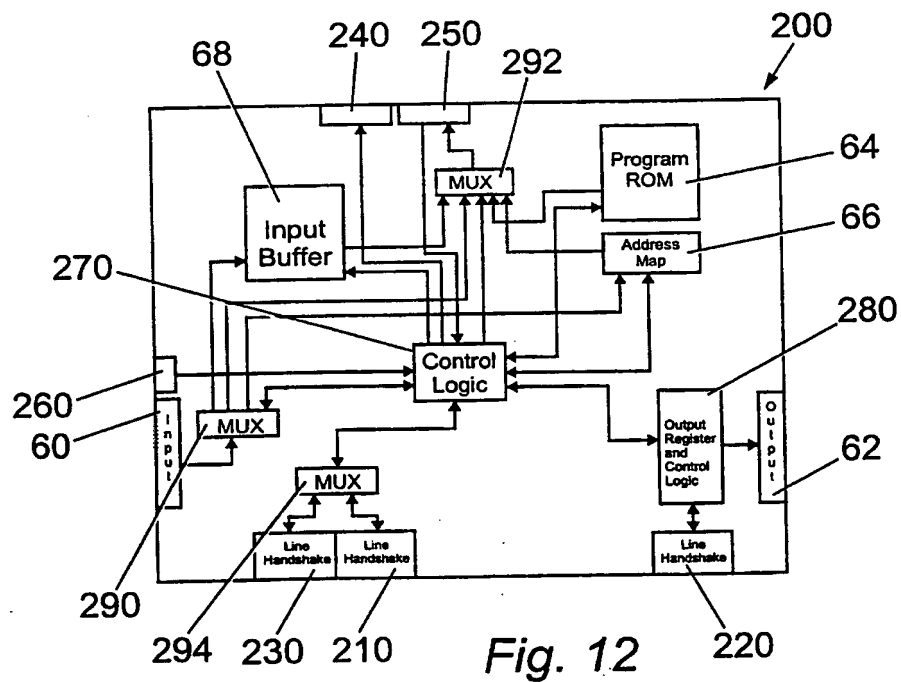
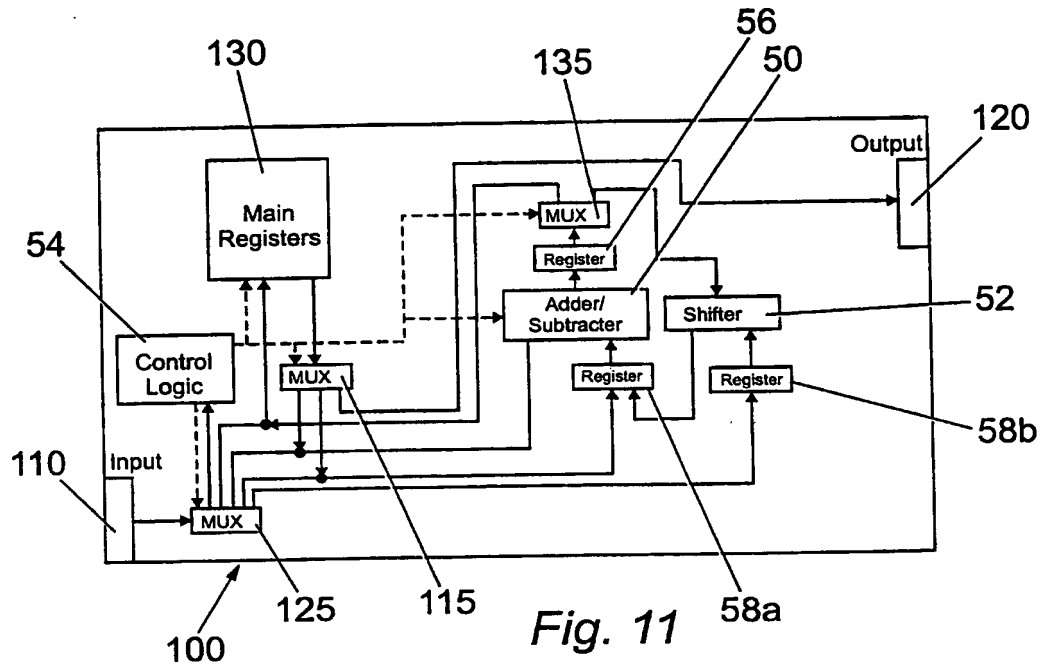


Fig. 8

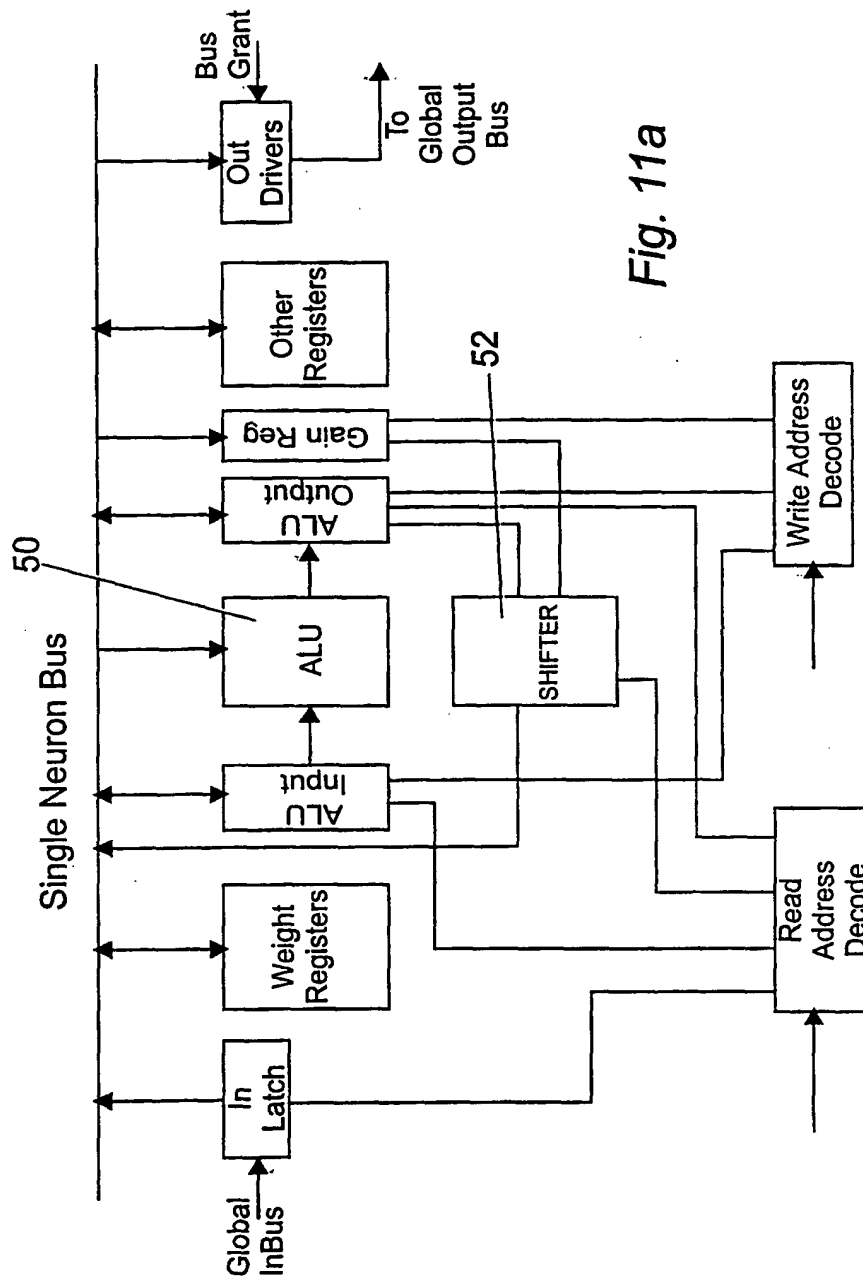
5 / 16

*Fig. 9**Fig. 10*

6 / 16



7 / 16



8/16

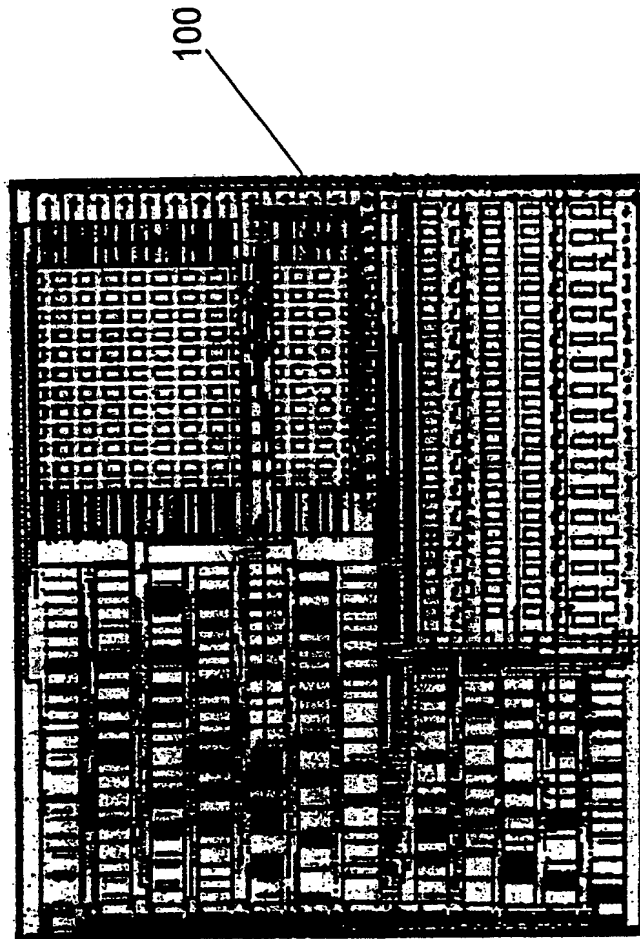


Fig. 11b

9 / 16

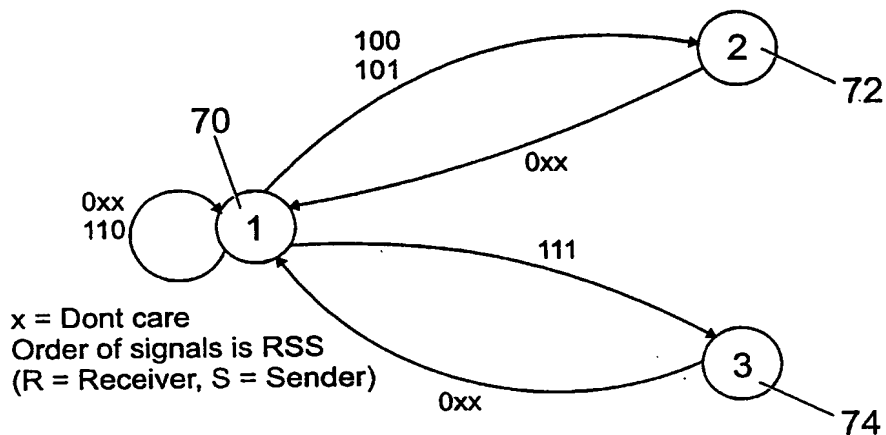


Fig. 13

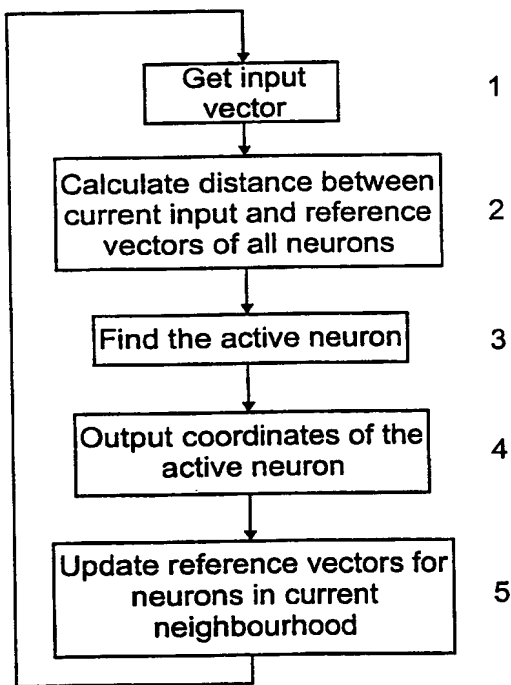


Fig. 14

10/16

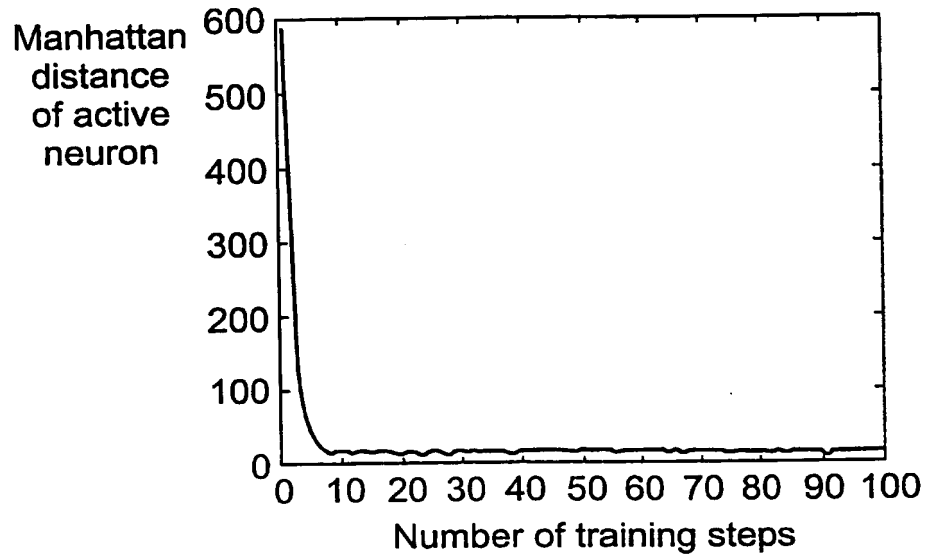


Fig. 15

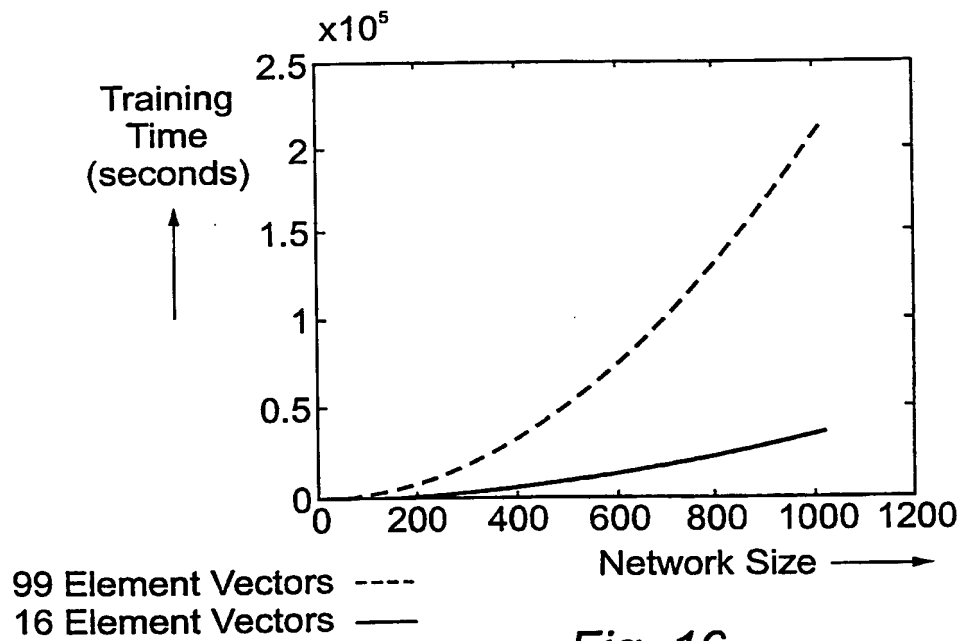


Fig. 16

11 / 16

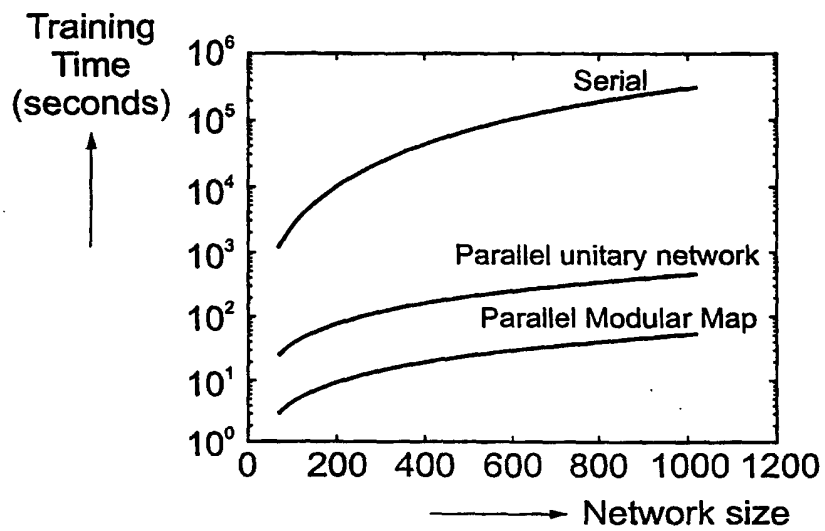


Fig. 17



Fig. 18

12 / 16

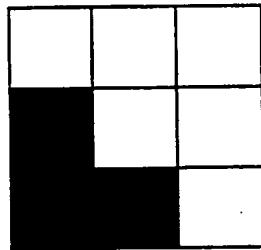


Fig. 19a

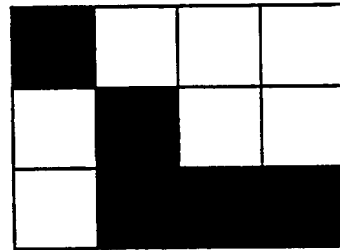


Fig. 19b

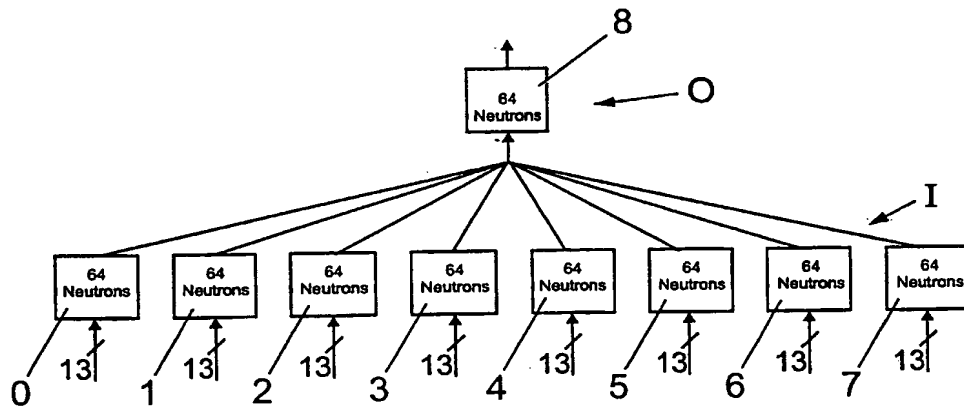


Fig. 20

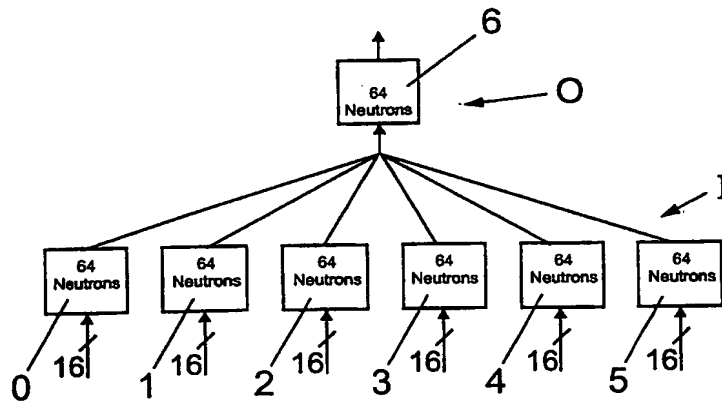


Fig. 21

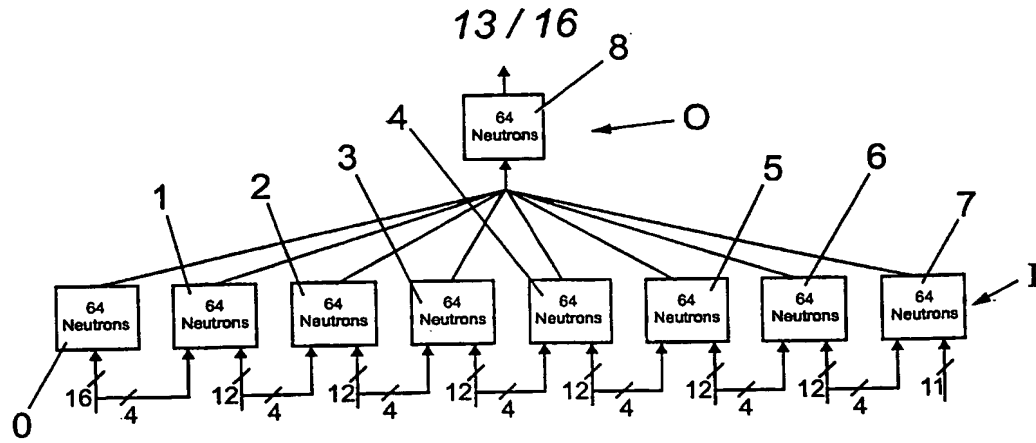


Fig. 22

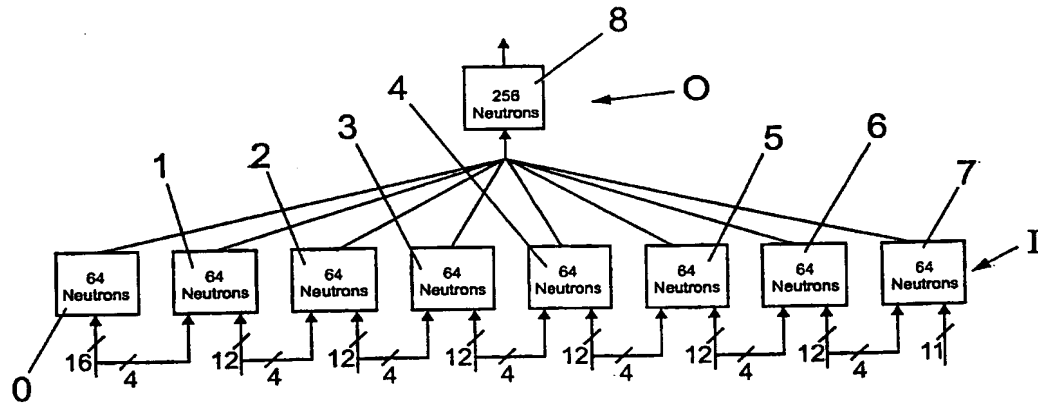


Fig. 23

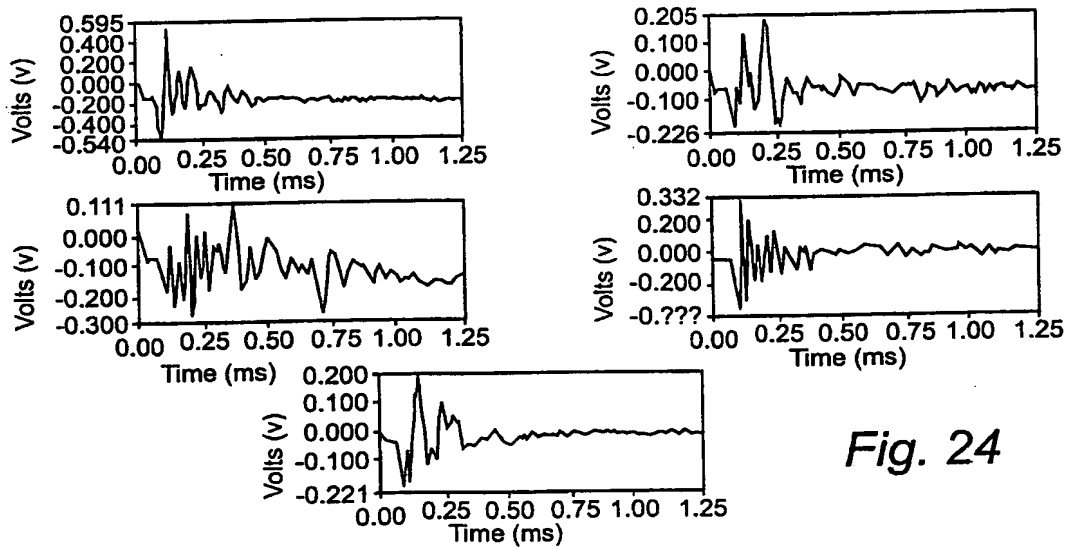


Fig. 24

14 / 16

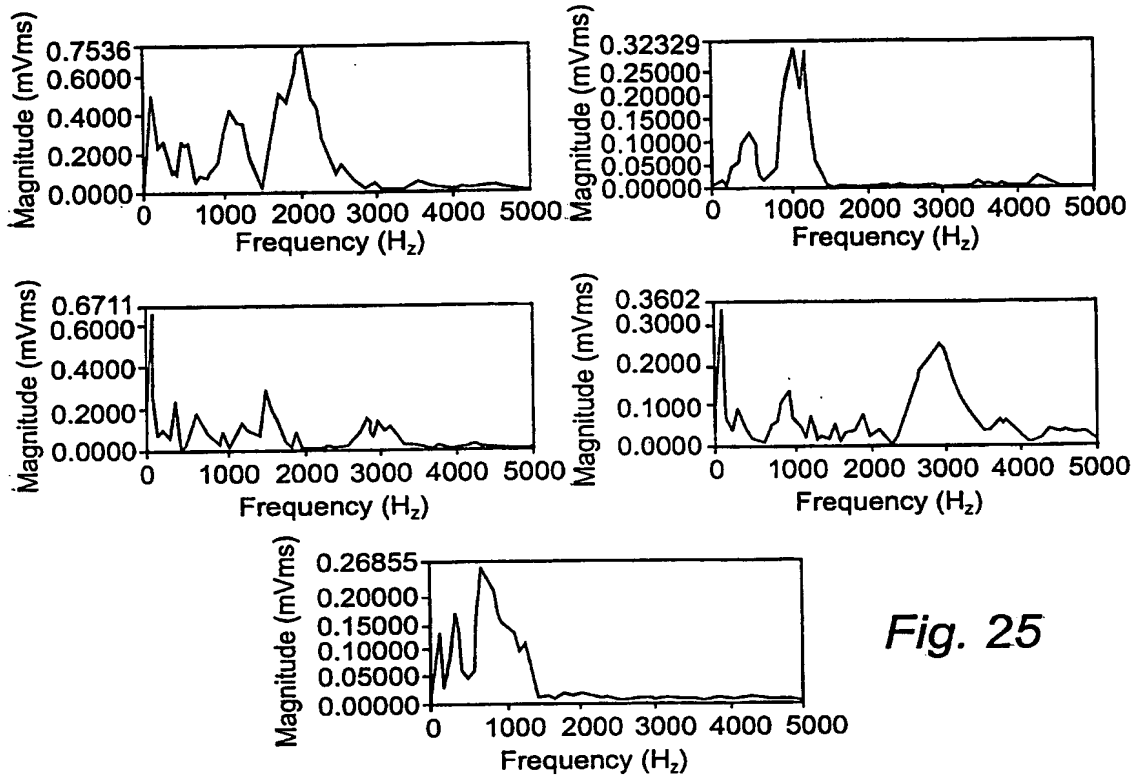


Fig. 25

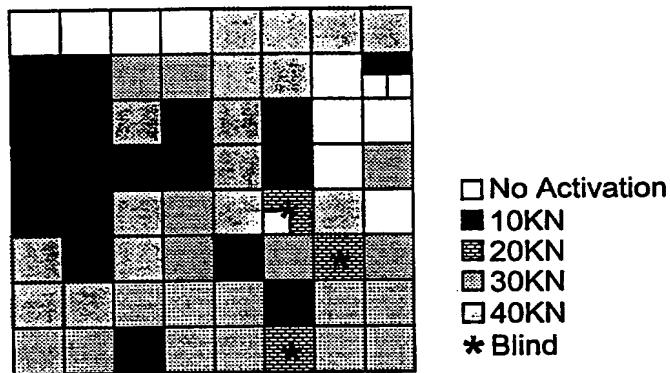


Fig. 26

Fig. 27

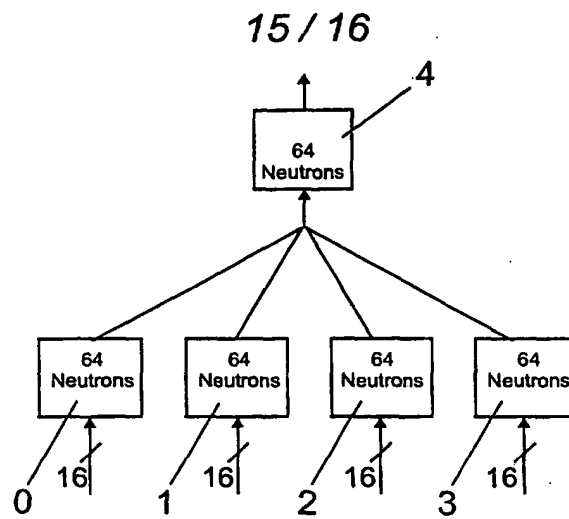


Fig. 28

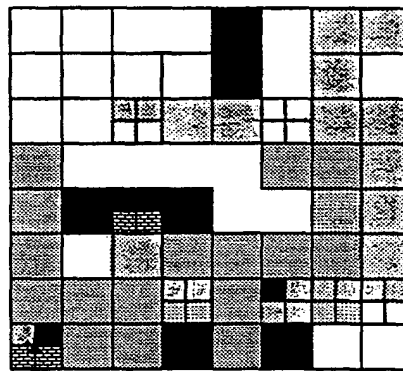


Fig. 29

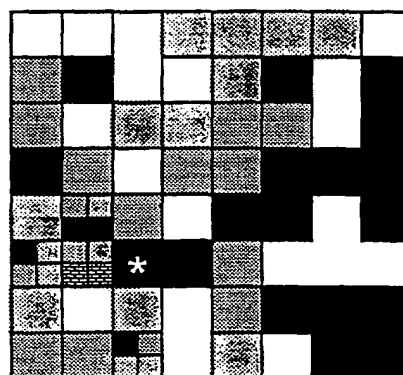


Fig. 32

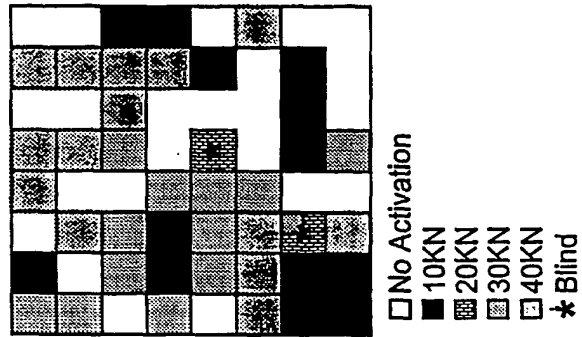


Fig. 31

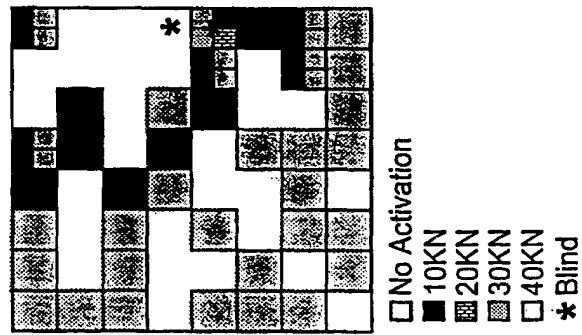


Fig. 30

